CONNECT – an architecture for a highly parallel system based on building blocks

Qiang Li and Naphtali Rishe* describe an approach to highly parallel architectures using 'self-sufficient' building blocks with limited communication power to build efficient large-scale parallel systems

Parallel processing and parallel architectures have become a promising solution to the ever-increasing demands for computational power. This paper describes an approach which uses 'self-sufficient' building blocks with limited communication power to build efficient large-scale parallel systems. The architecture, called CONNECT, provides a medium granularity parallel processing environment with the following properties: the parallelism is fairly high, the interprocessor communication pattern is random with both small messages and large data sets being sent between processors, and a substantial amount of computation is done by individual processors.

architectures **parallel systems** **hybrid networks**

Parallel processing and parallel architectures have become a promising solution to the ever-increasing demands for computational power. Numerous parallel architectures have been built or proposed. Each of these architectures is based on the philosophies, technologies and characteristics of the intended applications. Some of the architectures have made an important impact on the research field at different points in time, e.g., the Connection Machine[1], TRAC[2,3], CEDAR[4], Ultracomputer[5], MPP[6], PASM[7], Hyperswitch[8,9], Cosmic Cube[10], CHiP[11], Non-Von[12], IBM RP3[13], STARAN[14] etc. Various architectural frameworks have been studied and reported[3,15,16–18].

Many issues are involved in designing parallel architectures. Those raised most often are performance–cost relations, scales of parallelism, small *versus* large granularity, shared *versus* distributed memory, packet-switching *versus* circuit-switching, bus-oriented *versus* point-to-point connections, reliability and fault tolerance etc. Most of the issues boil down to the conflict between the interprocessor communication capacity of a system, the limitations imposed by the level of technology and the cost. Excellent analyses of these issues can be found in References 3 and 19.

This paper describes an approach which uses 'self-sufficient' building blocks with limited communication power to build efficient large-scale parallel systems. By 'self-sufficient' we mean that each building block is a complete computing unit. Each of the building blocks that we use has a fairly powerful processor, a local memory module and a limited number of point-to-point communication links. The data routing between links of a processor requires CPU intervention. The Inmos transputer[18] is a real-world model of these building blocks. As implied by the nature of the building blocks, the architecture is a distributed memory system.

It is generally agreed that parallel architectures are very application dependent and that it is difficult to have a true general-purpose parallel architecture. The parallel architectures based on building blocks have the advantage of being relatively easy to tailor to suit particular applications without involving VLSI chip design and processing. When properly designed, this type of architecture can be cost effective since the building blocks are often relatively cheap to produce. On the other hand, such building blocks have their limitations due to the fact that their design must be based on a set of assumptions and trade-offs so that they can be used in a wide range of applications.

Theoretically, it is easy to build a large-scale system with this type of building block since there is no direct contention for memory or communication links and no capacitive penalty as more processors are added to the network. However, such systems tend to have large

School of Engineering, Department of Computer Engineering, Santa Clara University, Santa Clara, CA 95053, USA
*School of Computer Science, Florida International University, Miami, FL 33199, USA
Paper received: 6 July 1991

0141-9331/92/020067-13

communication radii due to the limited connectivity of the individual processors. Further, since the communication related tasks and the non-communication related tasks share processors, they tend to interfere with each other. These impose a severe limit on the effective parallelism that can be built into a system. The problem becomes especially significant when the system works in an environment where the communication pattern is random and both small messages as well as large data sets are to be sent between processors.

The architecture described in this paper, called CONNECT, challenges the limitations imposed by the nature of the building blocks. The environment which CONNECT is intended to provide is a medium granularity parallel processing environment with the following properties: the parallelism is fairly high (in the range of a thousand); the interprocessor communication pattern is random, with both small messages and large data sets being sent between processors; and a substantial amount of computation is to be done by individual processors. The potential systems of such environments are parallel database machines, parallel control systems with many inlets and outlets, and interactive large-scale simulation systems. In addition to the building blocks described above, dynamically configurable passive switching components which are compatible with the point-to-point communication links are assumed to be available.

Like most large-scale multiprocessor systems, the heart of CONNECT is its interprocessor communication network. The goal of the network is to provide short and steady delivery times for short messages and high effective bandwidth for large data sets. The argument is that the short messages are often control messages such as synchronization signals, data locking and unlocking signals, acknowledgements etc. A short and steady delivery time for the control messages will have a direct impact on the overall system performance. We emphasize the steadiness of delivery time because a predictable response to the control messages will facilitate an efficient higher level design. On the other hand, a higher effective bandwidth implies a faster delivery time for the large data sets.

STRUCTURE OF THE INTERPROCESSOR NETWORK

Interprocessor networks are a sub-area of a more general area, namely, interconnection networks. Interconnection networks have been well studied and there is a rich collection of literature on the subject. Reference 20 is a collection of many important works. Chapter 5 of Reference 3 gives analysis of, and comparison between, many interconnection networks. A taxonomy of interconnection networks and graphs is also given in Reference 3.

Roughly speaking, interconnection networks can be divided into two categories: the statically connected packet-switching network and the dynamically configurable circuit-switching network, each with its own pros and cons.

The packet-switching network has the advantage of fast response time to the sending processors and the flexibility of being able to send messages to any node at any time asynchronously. It is suitable for short and frequent messages between processors. The main disadvantage of the packet-switching network is that when a large amount of data is sent, the packaging and depackaging time and the store-and-forward delay can be very long. In addition, interference between the large data package and the small messages makes the behaviour of the small messages unpredictable.

On the other hand, the circuit-switching network has the advantage of sending a large amount of data directly to the destination at the hardware speed without disturbing other nodes in the system. However, when the switching components are passive, i.e. the switching is done by external control signals instead of the messages themselves, which is a common case in the hardware environment that we assume, the circuit-switching networks suffer from the circuit set-up time delay, especially when short messages are sent.

To achieve our goal of satisfactory performance for both the short messages and the large data sets without excessive hardware cost, we have developed a hybrid of the packet-switching network and the circuit-switching network. Figure 1 shows the block structure of the interprocessor network, which consists of a packet-switching network, a circuit-switching network and a network of circuit control processors. A cost analysis of the network is given in a later section.

The basic idea is that when a short message is sent, it is delivered through the packet-switching network, which provides a quick delivery time; when a large data set is sent, a circuit setup request is sent to the circuit controllers through the packet-switching network, the controllers set up a dedicated circuit between the originator and the destination and the large data set can be sent directly to the destination. Furthermore, since the packet-switching network can have the luxury of sending only very small messages, the requirements of the design of the network and its components become less rigid, and packet-switching network behaviour related to the short messages becomes steady and predictable.

We now describe some details of different parts of the network. To clarify the discussion, we call the processors that do the actual data processing the 'data processors' and the processors dedicated to communication tasks (in the packet-switching network) the 'communication processors'.

The packet-switching network

The packet-switching network is a 'tree-shaped' network with the data processors as the leaves, as shown in Figure 2. To avoid a bottleneck, the 'root' of the tree is a small hypercube network instead of a single node. In other words, one can imagine a small hypercube with a subtree hanging from each of its nodes with the data processors at the bottom of the subtrees. The word 'subtree' is used in a loose sense, since the whole structure is not really a tree. As we will discuss later, the hypercube network is also the control network of the circuit-switching network. The size of the hypercube and the degree of the subtrees depend on the size of the entire system and the characteristics of the hardware components. The reason for selecting the tree-shaped network is threefold: a tree network has a reasonable communication radius; it is natural for the subtrees to concentrate the circuit setup requests from the data processors to the circuit control processors at the top of the subtrees; and the requirements of the connectivity of a tree node are flexible.

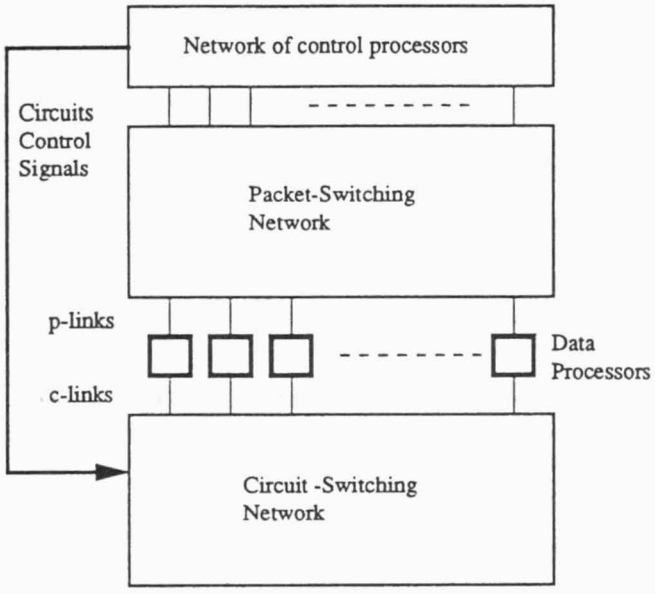

Figure 1. Block structure of CONNECT

The circuit-switching network

The circuit-switching network is a three-stage Clos network[21] as shown in Figure 3. Each box in the figure is an $n \times m$ crossbar switching component. There are three columns of switches: the input switches, the output switches and the intermediary switches. The links between the switches are uni-directional. The leftmost column is the output side of the data processors; the rightmost column is the input side of the data processors. For convenience, we call the links between the data processors and the switches, and between the different columns of switches, the 'input c-links', the 'output c-links', the 'input s-links' and the 'output s-links', respectively, as marked in Figure 3. A Clos network is denoted by $N(m, n, r)$ if its input, output and intermediary switches are $n \times m$, $m \times n$ and $r \times r$, respectively.

The three-stage Clos network has been selected because it can simultaneously connect many processors without too many layers of switching components, which could result in a long propagation time. Although the Clos network is known not to have a routing algorithm local to the switch components, it does not concern us since the routing is done by the external control in our case, due to the hardware constraints assumed.

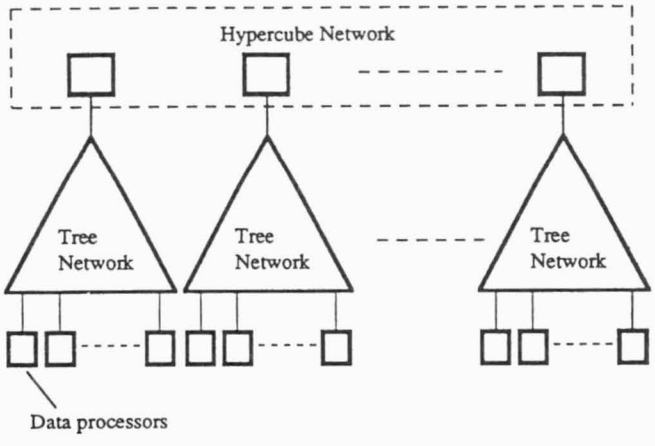

Figure 2. The tree-shaped packet-switching network

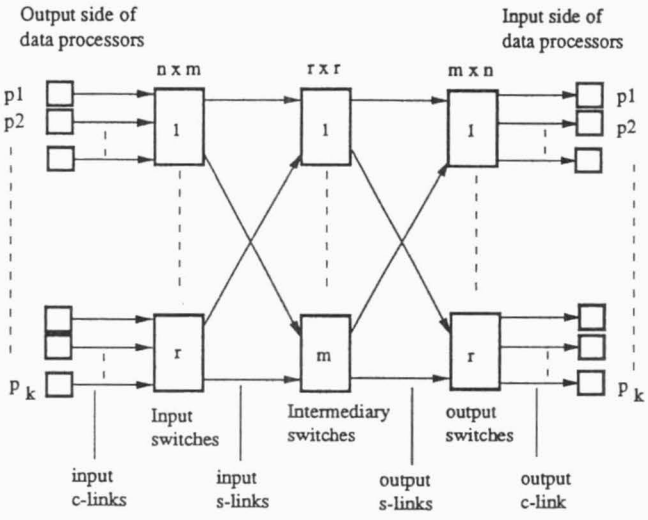

Figure 3. The three-stage Clos network

Two important properties of an $N(m, n, r)$ network are given below, and the detailed proofs can be found in References 21 and 22.

- If $m \geqslant n$, then for every partitioning of the set of all processors into pairs, there exist connection configurations where all the pairs talk simultaneously.
- When $m \geqslant 2n - 1$, the network is non-blocking, that is, there is always a path available between any idle input c-link and any idle output c-link, independent of the connection history.

The network of the control processors

The circuit controller of the circuit-switching network is responsible for selecting an available path for setting the circuit up, sending the hardware signals that actually set the circuit up, keeping track of the current status of the network, maintaining a queue of the connection requests that are unable to be satisfied for the time being, etc.

There are two problems which can prevent the control processors from achieving short circuit setup time and high circuit setup rate. First, the large amount of requests coming from the data processors have to converge to the controller, which presents a communication bottleneck. Second, the processing speed of the controllers must be high enough to handle the flow of requests. To alleviate this problem, a group of control processors is employed for the task, applying the parallel processing concept to the parallel architecture control itself. The circuit setup requests will arrive at one of the control processors depending on where the request is originated. A control processor receiving a request will process the request with the cooperation of the other control processors. In this way, the requests arrive through many independent channels and are processed in parallel by many processors. Thus the controller bottleneck can be significantly relaxed.

As mentioned above, the control processors are linked into a hypercube network which is also used as part of the packet-switching network. Each node of the hypercube is called an hnode. Figure 4 shows the relationship between the hnodes and the other components of the network.

The data processors at the bottom of the subtrees have not been drawn explicitly. Each switch is connected to a

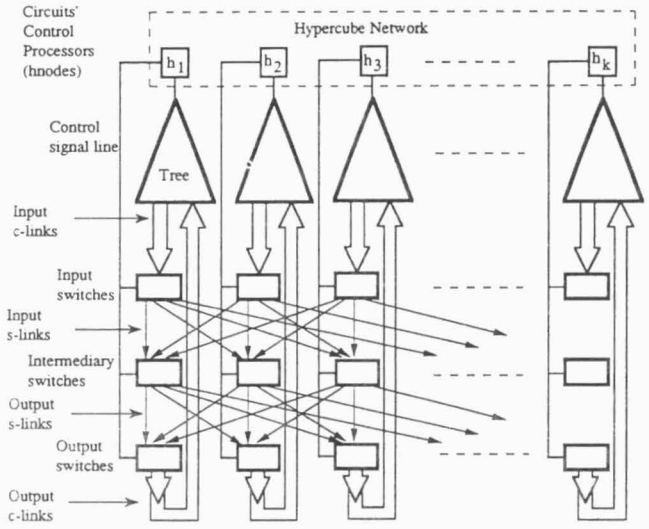

Figure 4. The connections between the hnodes and other parts of the network

set of data processors. We say that a switch is connected to a tree if the switch is connected to data processors in the tree. Each hnode controls the switches connected to its tree and one or more intermediary switches. For convenience, an intermediary switch was drawn together with each pair of input and output switches. In practice, the number of input and output switch pairs and the number of intermediary switches are often not the same.

ROUTING IN THE PACKET-SWITCHING NETWORK

Routing in the packet-switching network is quite straightforward. There is a system-wide numbering for the data processors. Each processor has a unique number and the leaves of any tree have consecutive numbers. The hnodes in the hypercube are numbered from 0 to $2^M - 1$, where M is the dimension of the hypercube. Since each hnode is the root of a tree, the trees are numbered accordingly. There is a static map between each tree-number and the range of data processor numbers in that tree. Each hnode has a copy of that map. The communication processors in the trees are called the tnodes. Each tnode controls d sub-trees, where d is the degree of the tree. The sub-trees are numbered from 1 to d.

There are two classes of messages travelling in the packet-switching network: data messages and control messages. Data messages are the messages passed between the data processors; all other messages are considered control messages. Both types of messages have the same header containing message-type, destination address and originator address, followed by the content of the message. Data messages are simply forwarded by the communication processors while the control messages are interpreted by the control processors.

The routing algorithm has two parts, one for the subtrees and another for the hypercube on top of the subtrees. The routing in the subtrees is straightforward. When a tnode receives a message, it will determine which way to forward the message by checking the destination address against a table and send the message accordingly. A message with a negative address number (in the case of a control message) is always sent to the parent. This process involves searching a table of a few entries, which can be done very quickly.

When an hnode receives a data message, it will send the message to the destination subtree according to its map. An hnode receiving a control message will process the message, as we will discuss in later sections.

The routing algorithm used in the hypercube is the 'fixed routing algorithm'. When the messages are short and the communication pattern is random, which is our case, the fixed routing algorithm is very efficient. In addition, a properly implemented fixed routing algorithm is deadlock free; a description of the deadlock-free implementation and a formal proof can be found in References 23 and 24. Furthermore, the tree network is inherently deadlock free. Therefore, the entire packet-switching network is deadlock free.

High performance of the packet-switching network is expected for the following reasons. First, only short messages are sent through the packet-switching network. Giving a bandwidth of the communication channels, the messages have less chance to run into each other and compete for a channel. Second, the network consists of dedicated communication processors, so the messages are processed promptly. Third, the packet-switching network is inherently deadlock free, so there is no deadlock prevention overhead.

In the following sections, we describe the control mechanism of the circuit-switching network, which is a more challenging problem: the parallel control of a Clos network.

PARALLEL CONTROL OF THE THREE-STAGE CLOS NETWORK

The status data and its representations

In this section, we describe the data structures and the information held in each hypercube node for the purpose of communication control. To clarify the discussion, a data processor is sometimes called a 'leaf' since it is a leaf of the subtree connected to the control node in question. Also, when we say 'to allocate a data processor', it means to allocate the link between the data processor and the Clos network.

Two logically independent types of information are kept by each control node: the status of the switch components of the Clos network which is under the control of the control node in question and the status of each data processor (leaf) under the control of the control node in question.

Status of the switching components

Each hnode keeps the following information about every switch under its control:

- InputVector: A bit vector representing the status of the input s-links. The bit positions in the vector correspond to the s-link numbers and a bit value of 1 indicates the corresponding input s-link is occupied.
- OutputVector: Similar to InputVector except that it represents the status of the output s-links.

- RelayTable: A table indicating the current connections between the input s-links and the output s-links on the intermediary switch.

An available path between an input switch and an output switch can be found using the InputVectors and the OutputVectors. Suppose the vectors are 8 bits long and a path is to be found between the input switch controlled by h_1 and the output switch controlled by h_2. InputVector$_{h_1}$ is 01001100, indicating that its links to the intermediary switches 2, 3 and 6 are occupied, and OutputVector$_{h_2}$ is 11001010, indicating that its links to the intermediary switches 1, 3, 6 and 7 are occupied. Let

$$V = \text{InputVector}_{h_1} \text{ OR } \text{OutputVector}_{h_2} = 11001110$$

Then the 0-bit positions of V are the intermediary switch numbers that are available. In this case, they are 0, 4 and 5.

Status of each of the data processors

An hnode keeps track of the status of each of its leaves. When considering the control algorithm, it should be clear that a leaf of an hnode is a logical entity represented by a set of data structures on the hnode. The following describes some of the data structures and the information that they represent. More details can be found in Reference 25.

- Status: Status of the leaf in terms of its connection to the Clos network. The status can be 'idle' or 'busy'. Their meanings are indicated by their names.
- WaitingQueue: Holds the Connection Requests at the hnode for the leaf in question. The requests can come from other data processors in the system or be initiated by the leaf itself.

The circuit control algorithm

We now describe the process of setting up a circuit. To simplify our discussion, let us assume the circuit-switching network is a non-blocking network, i.e., as long as the two processors to be connected are not currently connected to something else, there is a path available between them. We will discuss other cases later.

Suppose the circuit-setup request is originated by a data processor, orig, and the destination data processor is dest. Let h_o be the root of the tree that contains orig, and h_d be the root of the tree that contains dest. Note that orig, h_o, h_d and dest are all defined in terms of a particular circuit setup request. Figure 5 shows a flow diagram of the process of setting-up a circuit. The messages that flow between the hnodes for the purpose of setting-up a circuit have the general form:

MT	DA	OA	... Other information

where MT, DA and OA stand for message-type, destination address and originator address, respectively. Additional information is contained in the message depending on the message-type, which will be described individually.

For convenience, the leaves of an hnode are not addressed by their processor numbers, instead they are addressed by their leaf numbers. The leaf numbers are a sequence of unique numbers local to an hnode starting

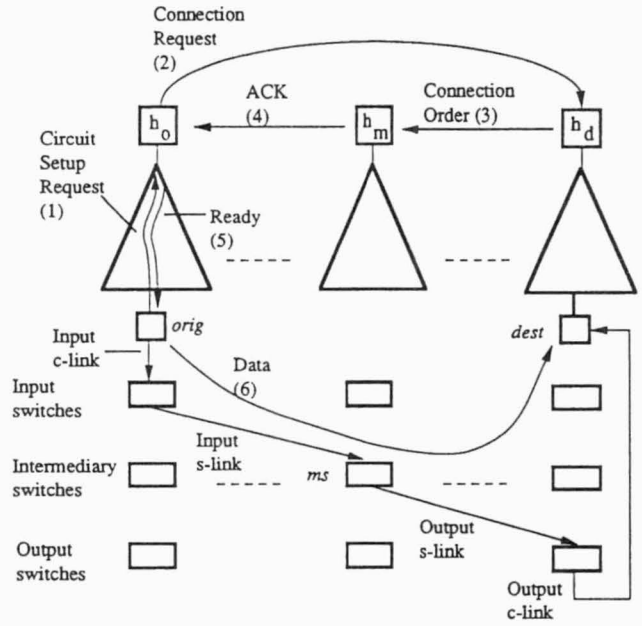

Figure 5. A process of setting-up a circuit

from 0. The leaf number of a leaf (data processor) under a particular hnode can be easily calculated from its processor number under a given system configuration. Therefore, the addresses of a leaf contained in the messages flowing in the control hypercube normally consist two parts: the number of the hypercube node which controls the leaf called cube number and the leaf number.

To make it easier to understand, we shall describe a process of setting up one connection circuit. Figure 5 will serve as a graphical aid for the discussion.

The process is described by the following numbered steps and the numbers are consistent with the numbers marked in Figure 5.

(1) Starting from the leaf orig, a circuit-setup request message (CSR) will be sent up the tree to the control hypercube node h_o.

(2) Upon receiving the CSR, a 'connection request' (CR) message will be constructed; if the status of the originator leaf is busy, the connecting process will be suspended by putting the CR message into the WaitingQueue and h_o will continue to handle other messages. The suspended process will resume when the CR is taken out of the queue. If the status of the leaf is idle, h_o will set it to busy, and determine the position of h_d in the hypercube, and send the CR to h_d. The connection request has the following format:

MT (CR)	DA (h_d, l_d)	OA (h_o, l_o)	InputVector of h_o

(This means that the message type MT is 'CR', the destination address field DA contains the pair (h_d, l_d), and so on.)

(3) Upon receiving the CR, if the status of the destination leaf is busy, the process will be suspended by putting the CR message into WaitingQueue of the leaf and the control node h_d is released to handle other messages. The process resumes from this point when the CR is eventually taken out of WaitingQueue.

If the status of the destination leaf is idle, h_d will set it to busy and check the InputVector of h_o against its own OutputVector to find an intermediary switch which has free links to connect the switch controlled by h_o to the one by h_d. Let *ms* indicate the intermediary switch selected by h_d and h_m indicate the hnode which controls *ms*. A 'connection order' (CO) is sent from h_d to h_m. The connection order has the following format:

MT (CO)	DA (h_m)	OA (h_d, l_d)
Switch Number (*ms*)	Request Originator (h_o, l_o)	

In the meantime, hardware signals are sent to the output switch to connect the output c-link and the output s-link connected to *ms*. Notice that h_o, h_d and h_m are not necessarily distinct, e.g. they can actually be the same hnode.

(4) Upon receiving the connection order, h_m will verify the validity of the connection by checking into its RelayTable. If the connection is valid, i.e. the links needed by the connection are not already occupied, then hardware signals will be sent to *ms* to connect the input s-link and the output s-link. Meanwhile, an 'ack message' (ACK) is sent to h_o indicating that the requested circuit has been set up. It is possible for a connection to be invalid when the CO message arrives. This is called a 'race condition' and we will discuss this case in a later section.

The ACK message has the following format:

MT (ACK)	DA (h_o, l_o)	OA (h_m)
Switch number (*ms*)	Request Destination (h_d, l_d)	

(5) Having received the ACK message, h_o will send signals to the input switch to connect the input c-link to the input s-link which links to the intermediary switch indicated by *ms*. A 'ready message' (READY) is then sent to the originating data processor, orig. In the meantime, the InputVector is modified.

(6) Upon receiving the acknowledgement (READY) from the control processor network, the data processor orig will send its data through the circuit, which effectively initiates the communications. Thereafter, the two connected data processors can communicate in any way they choose.

(7) When orig decides to release the circuit, it sends an 'originator disconnect circuit request' (ODCR) to its control node h_o.

(8) h_o will in turn send an 'intermediate disconnect circuit request' (IDCR) to h_m. The InputVector of h_o will be modified to release the related s-link. Also, if WaitingQueue is empty at this point, the status of the leaf is set to idle. Otherwise, the first CR message will be taken out of the queue and the connection process for that CR message will be continued. The IDCR have the following format:

MT (IDCR)	DA (h_m)	OA (h_o, l_o)
Switch number (*ms*)	Request Destination (h_d, l_d)	

(9) Upon receiving an IDCR message, h_m will send a 'destination disconnect circuit request' (DDCR) to h_d and the RelayTable of h_m will be modified to release the related links. The DDCR message has the following format:

MT (DDCR)	DA (h_d, l_d)
Switch number (*ms*)	Request originator (h_o, l_o)

(10) Upon receiving a DDCR message, h_d will modify its OutputVector to release the related link. If WaitingQueue is empty, the status of the leaf will be set to idle. Otherwise, the first CR message in the queue is activated.

Many such processes can be carried out simultaneously. They are independent of each other except when two connections need the same processor, in which case one of them has to wait.

In the following sections, we discuss some details related to the control algorithm.

Deadlock and its elimination

Two data processors are involved in setting up a connection circuit, and the c-links to both processors have to be available before the connection can be established. In general, they are not always available at the same time. Therefore, we have to allocate (hold) the one that becomes available first and wait for the other. Since the connection patterns and times are random, there can be a hold-and-wait cycle as shown in Figure 6. Since the connections are non-preemptive and the links are non-sharable between connections, Figure 6 actually shows a deadlock situation.

To solve the problem, the algorithm is designed so that one always allocates the data processor with a small id number and waits for the one with a greater id number. In this way, it is impossible to have a circular waiting situation, and so the deadlock is prevented. The implementation of the solution is that when a connection is from a leaf of a smaller number to a leaf of a greater number, the connection request is marked as an 'up-going' CR. An up-going CR will allocate the originating leaf on h_o first by setting its status to busy and then go to h_d to wait for the destination leaf. On the other hand, a non-up-going CR will go to h_d first to allocate the destination leaf and then the process will wait for the originating leaf when the ACK message arrives at h_o. In either way, the leaf with a smaller number is allocated first. The proof of the correctness of this solution is trivial.

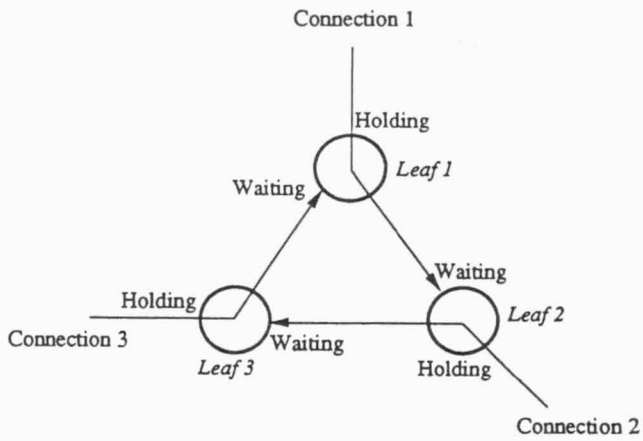

Figure 6. A deadlock situation

Selection of the intermediary control mode

As described earlier, three control hypercube nodes have to be involved to establish one connection, namely, h_o, h_d and h_m. The h_o and h_d are decided by the location of the originator and the destination, but h_m has to be selected by the control algorithm. The selection strategy can have direct impact on the speed of the circuit-setup process. The basic goal is to minimize the distance from h_d to h_o through h_m.

The problem can be generalized to a general hypercube allocation problem as the following. Let $d(n_1, n_2, n_3)$ be the shortest distance from n_1 to n_2 through n_3 and $f(n_1, n_2, n_3)$ be a Boolean function which yields a true value if the three arguments meet certain conditions. Given two nodes in the hypercube, n_1 and n_2, find the third node, n_3, so that $d(n_1, n_2, n_3)$ is minimal and $f(n_1, n_2, n_3)$ is true.

A selection strategy and an algorithm implementing the strategy, which is similar to the feedback shift register technique, have been developed[26]. A simulation has shown that utilizing the strategy can reduce the traffic in the control hypercube by 20% compared to the case where the selection is done in the ascending order of the id numbers of the nodes starting from h_d.

Race conditions

In some situations, a problem can occur in the above algorithm: h_o could send a connection request with an outdated InputVector if there is any connection request previously sent from h_o which has not been acknowledged. Therefore, the h_ds of two different connections originated from the same h_o could select the same intermediary switch, resulting in a conflict over the input s-link which connects h_o to the selected intermediary switch. This is called the 'race condition' and is illustrated in Figure 7.

The situation is resolved as follows. If h_m detects that a new connection requires a link which is already in use by checking its RelayTable, a CANCEL message will be sent back to h_d, and h_d will try to find another available intermediary switch. The node h_d also counts how many CANCEL messages have been received for a particular connection. If the count exceeds a limit, say three, the InputVector of h_o of the connection is most likely too outdated, and a RESEND message will be sent back to h_o of the connection and h_o will start the whole process over again with the current value of its InputVector.

This solution requires some extra traffic in the system and some overhead time for the connection involved. However, a simulation has shown that the probability of the race condition occurring is very low (less than 0.001 with the assumed communication pattern.) Therefore the above activity happens infrequently, and thus the total overhead is negligible.

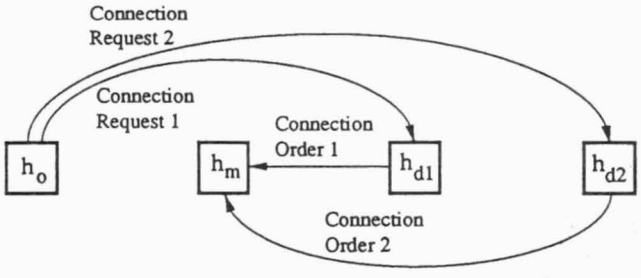

Figure 7. An occurrence of the race condition

When a connection is blocked

In general, the Clos network can be a blocking network. For example, it is often the case that the switch component has the same number of input and output links, say x links on each side. The most efficient Clos network that can be built based on the switch components is $N(x, x, x)$, which is a blocking network. That is, it could happen that two idle processors cannot find a path between them. A simulation on an $N(32, 32, 32)$ Clos network shows that under a random connection pattern, the chance of the blocking situation occurring is very low (less than 0.0001.) Therefore, the blocking situation can be simply resolved by a timed re-try or wait until a connection from the h_o is relinquished without creating much delay and traffic.

ENHANCEMENT TO THE DESIGN

Several modifications have been made to the original design to enhance the architecture. The enhancements are made mainly based on the consideration of performance and the reliability of the system.

Separation of the control hypercube from the packet-switching network

The hypercube on the top of the subtrees is the critical part of the system. Its speed has a direct impact on the performance of both the packet-switching network and the circuit-switching network. Also, when a hypercube node fails, it is difficult to keep the system alive without losing performance significantly. To avoid the problem, another hypercube is added to the system which is connected to the system exactly as the original one. Figure 8 shows the connections. Since the nodes of the hypercube are a small portion of the processors in the system, the extra hardware cost is not high. Under normal operational conditions, one hypercube, called the control hypercube, controls the Clos network, and the other,

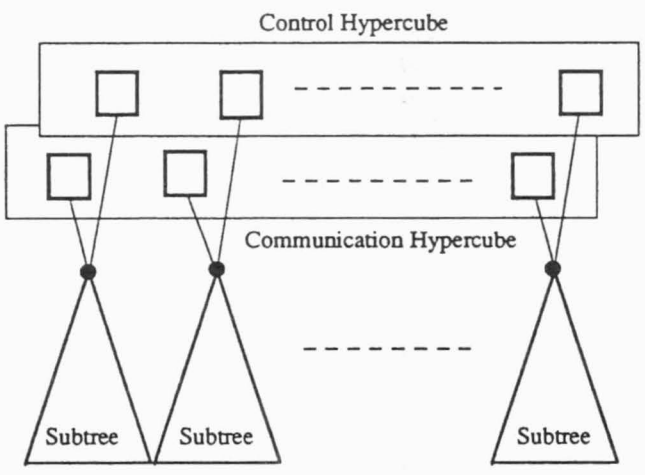

Figure 8. Using two hypercubes

called the communication hypercube, is in charge of passing messages between the subtrees.

Since less traffic flows through each hypercube, the maximum throughput of the packet switching network is improved. Also, the speed of the circuit setup is increased and is less sensitive to the traffic load of the packet-switching network.

Furthermore, fault tolerance of the system will be improved by using the two disjoint hypercubes. We will discuss this issue later.

Data processors sharing circuit-switching network links

The data processors are partitioned into pairs and the two processors in a pair are connected through one link. This requires each data processor to have one more link, which is easy to satisfy since the requirement for the connectivity of the data processors is very low.

There are two major functions of the additional connection. First, it provides an alternative channel for a data processor through the circuit-switching network. Figure 9 shows an example. Processors *a* and *b* are partners. When processor *a* is connected to processor *c* through the network, another connection to processor *a*, say from processor *d*, can be made to processor *b* and the connection can be relayed through processor *b* if *b* is not connected to any one at the time. Since there is only one step relay and the data can be transmitted in small packets, the store-and-forward delay is expected to be low. There is a computation overhead for the relaying processor. However, since a processor only relays for one other processor, the overhead is low. Statistically, when a totally random communication pattern is present, the benefit of using the additional connection is not significant. A simulation shows that the additional connection increases the effective bandwidth of the system under a random communication pattern by 15%. The additional connection plays a more important role when processor *a* is connected to another processor for the transmission of a very large quantity of data; the alternative route will enable processor *a* to transmit large data sets to the other processors simultaneously. When the higher level application can be designed in a way that the two processors are closely logically related, which is often possible, the additional link can carry a significant portion of the traffic. In addition, the additional link can provide a local small granularity parallelism without too much additional hardware.

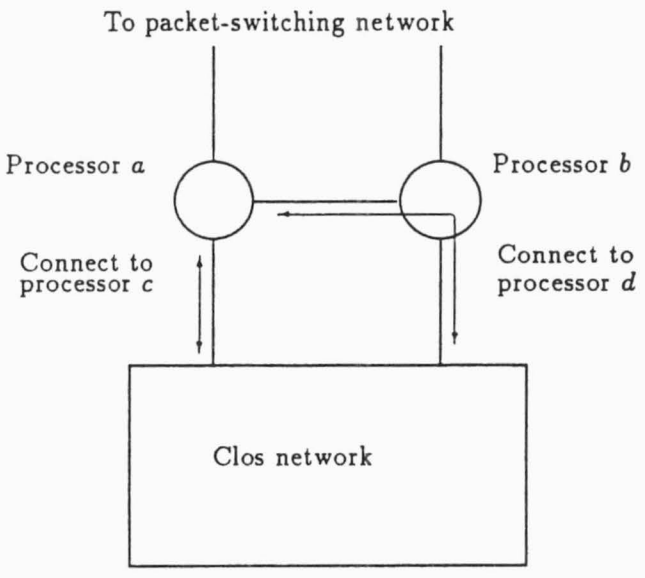

Figure 9. The additional link between data processors

The second function of the additional link is related to the improvement of reliability, which will be discussed later.

FAULT TOLERANCE

Hardware redundancy

Like most systems with built-in redundancy, CONNECT also provides high reliability. We now analyse several parts of the system.

In the enhanced packet-switching network, there are two hypercubes on the top of the subtrees and they are totally symmetric and disjoint. When one of the hypercubes is down, e.g. a node of the control hypercube malfunctions, the function of the failed hypercube can be easily switched to another hypercube without losing performance significantly.

Also, in the enhanced packet-switching network, additional connection is provided between data processors. The processors can be partitioned so that there is always an additional link between the boundary of two subtrees. In case the root of the subtree fails, the messages can still flow out of the subtree through the additional links to other subtrees and then climb to the top of the subtrees if necessary. Thus, the failure of one of the roots will not stop the system. In terms of performance under such circumstances, it should be acceptable bearing in mind that only small messages flow through those links. A drawback of this is that since the processors on the boundary of the subtrees are not topologically symmetric to the roots of the subtrees and the routing through the connection between data processors is not part of the normal routing algorithm, the software solution will not come naturally and will slow down a part of the system. Nevertheless, the system can be kept alive.

Furthermore, if the hardware permits, all the data processors can be linked into a big ring, which is similar to the X-tree[27]. In that case, the routing in the emergency situation is quite straightforward. A message has simply to go through the ring until it gets out of the subtree with the troubled root.

We now discuss the situation of the circuit-switching network, when a path of the circuit-switching network fails, i.e., a data processor finds itself unable to send a message through the circuit-switching network. There are two things it can do. First, an alternative connection can be established through a neighbour and the messages can be relayed to the destination. Second, the messages can be sent through the packet-switching network for the time being. Although the performance under these circumstances will be worse than normal, the system can be kept alive until the problem is fixed.

If an intermediary switch component fails, the effect to the system is almost negligible since there are many of them and they are all symmetric. On the other hand, if an input or output switch component fails, the situation will be worse. The data processors connected to the failed switch will all have to divert their long messages through

the packet-switching network, which could cause a significant system-wide slow down. Thus, the input and output switch components are the weak spots of the architecture.

In summary, the system can stay alive when any one of the components fails, though the performance may suffer. The difference between the effects of different failures lies in the amount of performance reduction caused by the failure.

Monitoring the system

Several data processors can be assigned the duty of monitoring the system. Each processor of the system can send messages to the monitors periodically to indicate that it is functioning normally. When a monitor notices unexpected silence from some nodes or subtrees that may have got disconnected from the packet-switching network, the monitor will send a 'how are you doing?' message through the packet-switching network to such node(s). If the node(s) fails to respond for a timeout period, the monitor will set up a temporary circuit to the node(s) through which the status of the node(s) can be obtained and the data from the node(s) can be relayed by the monitors.

PERFORMANCE EVALUATION OF CONNECT

To have a realistic performance evaluation, the Inmos transputer is used as the hardware model of the building blocks. The configuration under investigation has 1216 processors, 960 data processors and 256 communication processors. It is assumed that the data processors have four communication links and the communication processors have seven communication links.

The circuit-switching network is a three-stage Clos network, $N(32, 32, 32)$, consisting of 96 Inmos C004 dynamic reconfigurable switches (32×32).

All the communication links are serial links, including those in the circuit-switching network, and have a speed of 20 Mbit s^{-1}. The experiments show that the effective one-directional transfer rate is 0.8 Mbyte s^{-1}. The links to the T800 transputer processors have a much higher bandwidth due to overlapping of data transmission and acknowledgement, but we choose to use a conservative number.

A simulation has been developed to evaluate the dynamic performance of the interprocessor communication network. The enhanced version, with two separate hypercubes on the top of the subtrees, is used. Figure 10 shows the connections of a subtree. The root has five subtrees and each of the second level nodes has six children. Each subtree has a total of 30 data processors. The hypercubes are five-dimensional, so there are 32 such subtrees.

The circuit-switching network can provide a link between any pair of data processors. Depending on the dynamic connection sequence, the number of pairs which can be linked simultaneously varies from 240 (480 processors) to the best case of 480 (all the 960 processors). Translated into the data transfer rate, it is from 384 Mbyte s^{-1} to 768 Mbyte s^{-1}. Bearing in mind that this is a static figure, the statistical dynamic performance is

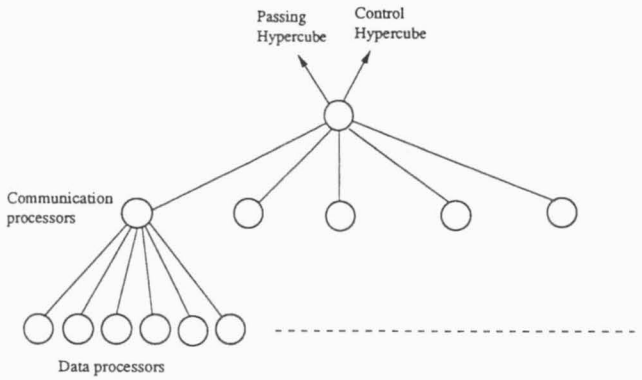

Figure 10. A subtree of the simulated system

expected to be much lower than the figures indicated here due to competition for the same nodes.

The capacity of the packet-switching network depends on the roots of the subtrees. Theoretically, the throughput is $32 * 0.8 = 25.6$ Mbyte s^{-1}. In practice, the throughput has to be kept much lower than the full capacity, since the average data transfer time will increase dramatically when the system reaches its full capacity - due to the channel contention. The following are some important assumptions made by the simulation.

- The communication links are assumed to be driven by DMA, i.e. all links in both directions can operate simultaneously.
- The message routing between the links on a processor is done by the CPU. The time taken by the CPU to route a message is assumed to be a uniform random number with an average of 30 μs.
- The links connected to a processor share a common 32-bit bus with the processor. The effect of the contention is implemented as an overall reduction of the link speed by a factor of 0.83; bearing in mind the speed of the bus is 20 to 40 times faster than that of the links, this is a conservative assumption.

In addition to the hardware system assumptions, the following assumptions are made for the data flow patterns:

(1) The interval time between two messages initiated by a data processor is a random variable with an exponential distribution.

(2) The message length follows a distribution with the density function

$$f(x) = 0.6 \frac{1}{\sigma x \sqrt{2\pi}} e^{-(\ln x - \mu)/2\sigma^2} + 0.4\lambda e^{-\lambda x}$$

which is a combination of a log-normal distribution and an exponential distribution. With this distribution, 60% of the messages are less than 100 bytes long and have a near log-normal distribution with an average message length of 50 bytes, and the messages which are more than 100 bytes long have a near exponential distribution with an average length of about 3000 bytes. This message size distribution was derived from an application of CONNECT to a parallel database machine[25].

(3) The destinations of the messages from a data processor are uniformly distributed.

Under the given distributions, Figures 11a and 11b show the relation between the throughput and the

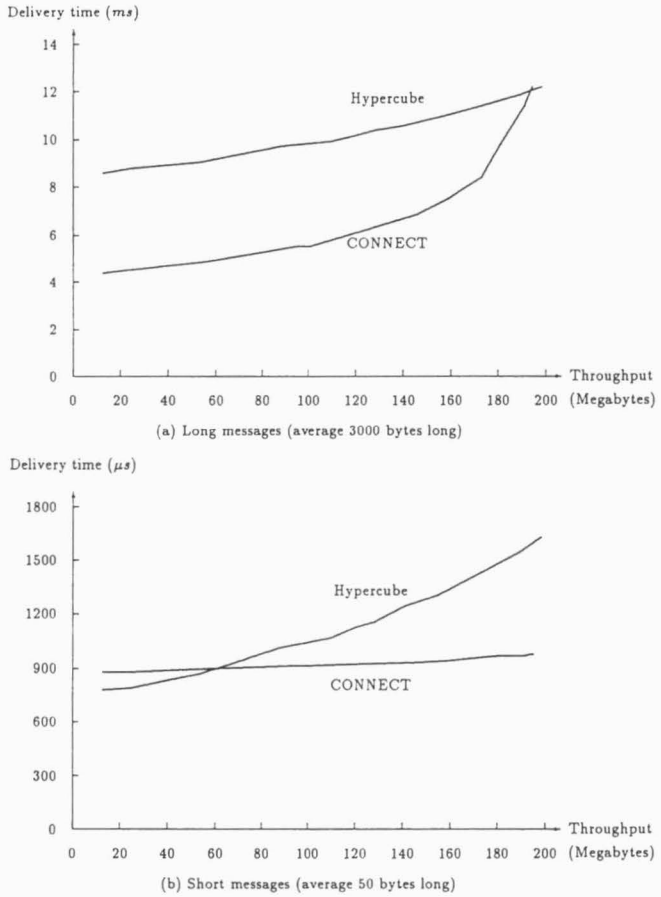

Figure 11. Throughput versus delivery time

message delivery time for the long messages and the short messages, respectively. The curves marked with 'hypercube' will be discussed later.

In the case of long messages, the delivery time climbs quickly after the system load exceeds 170 Mbyte s^{-1}. Since the bandwidth of the circuit-switching network is a constant once a circuit is set up, the curve in Figure 11a mainly reflects the change of the circuit setup time. Notice that the increase of circuit setup time is not due to the falling of the speed of the circuit setup network, but mainly because of the dynamic property of the Clos network. When the system load increases, the waiting time for the connection of a data processor to become available increases in a Clos network. Under the given distributions, the system shows a practical throughput limit of 200 Mbyte s^{-1}.

For short messages, the delivery time is from a little less than 900 μs to 970 μs throughout the throughput range of a few Mbytes to 200 Mbyte. This is the deliberate result of one of our design goals. The short messages are mostly control messages such as synchronization signals, data locking and unlocking signals, acknowledgements etc. The delivery speed of those messages has a direct impact on the system performance. Therefore, we want them to be as small as possible and as steady as possible. To show the steadiness, an important parameter that we are looking for is the correlation between the size of messages and their delivery time. A close correlation between the sizes of the messages and their delivery times indicates that the message size and its travel time have a nearly linear relation and the delivery time for a given length of messages has a small standard derivation. This provides stable and predictable behaviour for the short messages, which is important in the design of efficient higher level applications. The simulation results indicate that short messages have a very high (message size)-(delivery time) correlation, i.e. 0.97 throughout the throughput range of 6 to 200 Mbytes. In Figure 12a the sizes and delivery times of about 1000 short messages from the simulation results are plotted.

In an attempt to have a performance comparison with other architectures, we select the hypercube architecture as a benchmark since it is well accepted as a very strong multiprocessor architecture framework.

A simulation was implemented for a hypothetical 10-dimensional hypercube system with 1024 processors, close to the number of data processors in the case simulated for CONNECT.

Most of the simulation assumptions are the same as for CONNECT. The processors are assumed to have the same characteristics as transputers except that there are 11 communication links on each processor, which is more than what we need in CONNECT.

The messages are passed in the hypercube in a store-and-forward fashion using node buffers. The E-cube routing with a route-shortest-first adaptive routing algorithm is used.

The processors in the hypercube cannot be assumed to be dedicated communication processors. However, the load of the non-communication related computation on each of the hypercube processors is assumed to be very low, i.e. 17% of CPU utilization. Choosing this utilization, we are comparing CONNECT to the near best case of the hypercube in question.

Notice that the comparison described here is far from exhaustive. There are many different types of hypercube architectures[8, 9, 17]. Comparisons to other architectures

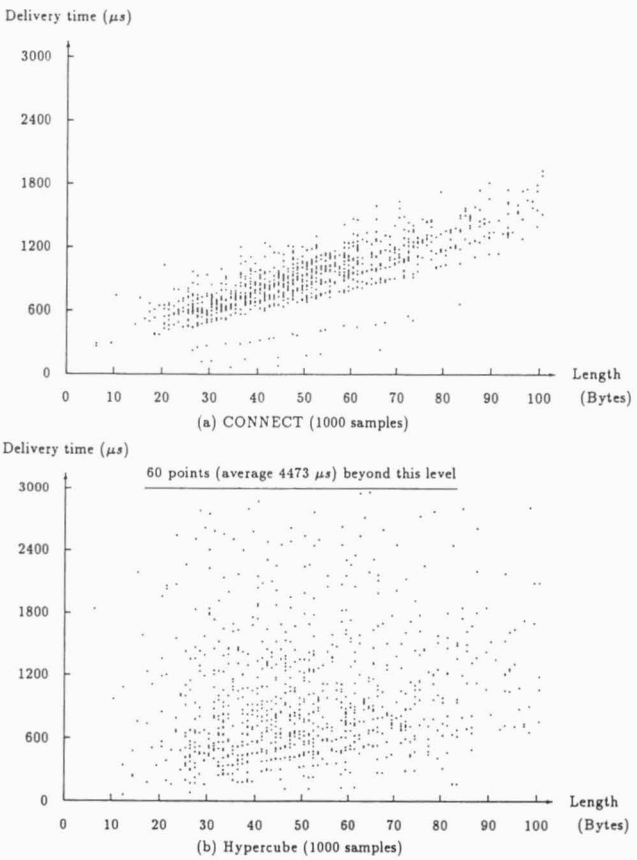

Figure 12. Sample message delivery times

may also be useful. More extensive comparisons in different perspectives should be conducted.

Figures 11a and 11b show comparisons of the (throughput)-(delivery time) between CONNECT and the hypercube in the case of short and long messages, respectively. Under a system load of 120 Mbyte s^{-1}, the average delivery time of CONNECT is 40% less than that of the hypercube for long messages, and 21% less for short messages. For short messages, the delivery times of the systems are very similar. However, the hypercube is more sensitive to the system load. Also, it is sensitive to the CPU load for non-communication related computation, which is not shown in the diagrams. The sensitivity is caused by the interference between the long messages and the short messages, which also causes the problem of non-stability of the delivery time for short messages, as we will discuss below.

Figures 12a and 12b depict 1000 short messages on the plain of (message size)-(delivery time) for the hypercube and CONNECT, respectively. The data were obtained under a system load of 120 Mbyte s^{-1}. It is clear that CONNECT has a much more stable delivery time. Linear regression analysis between message size and delivery time shows a mean square error of 28 816 μs^2 for CONNECT and 1 789 561 μs^2 for the hypercube; the latter is 62 times higher. The reason is that in CONNECT the short messages and long messages travel through different parts of the network, and there is very little interference between them, which is not the case for the hypercube. In fact, for any packet-switching network in which both long messages and short messages travel, the (message size)-(delivery time) correlation for the short messages is expected to be low and to grow worse when the system traffic load increases.

DISCUSSION ON COST

Since CONNECT uses a hybrid network, it seems to be more complex than a single network system. A cost analysis is needed. In general, cost analysis is a difficult issue which involves many factors such as overall design criteria, constraints etc.

In a circuit-switching network comprising active switching components, the circuit is set up by individual switching components according to the routing information contained in the message, and there is no need for an external control mechanism. Examples are the Hyperswitch of NASA/JPL[8, 9] and the Wormhole network[17].

However, in our design, passive switching components (the connections are set by external control signals) are assumed for the following reasons. To avoid long propagation time, we want the number of stages of the circuit-switching network to be as small as possible as in the Clos network which can connect large numbers of processors with a small number of stages (three stages in our case.) However, the Clos network does not have a routing algorithm local to each switch component. Hence, it is difficult to build routing logic into the switch components. Furthermore, since the overall design is based on building blocks, it is desirable to use general purpose programmable switches controlled by external signals. Therefore, an external control mechanism is needed, a control network in our case.

Besides the small number of network stages, we gain other advantages by using two networks. By sending short and long messages through different networks, the performance is improved, especially the stability of short messages. Also, the existence of two networks provides a higher reliability.

While it is difficult to have absolute figures to judge the cost or compare to other systems, we make the following observations.

- Since the Clos network is a passive network, the switching components are much simpler than those used in an active network. Thus, the cost of the switching components should be much lower.
- The requirement of the packet-switching network is less than that of a regular full scale network, since it only passes short messages. The network is simplified significantly compared to a full scale network. This implies a much lower cost.
- In terms of communication lines, for a configuration of 1024 processors, there are 4096 unidirectional single communication lines in the Clos network and about 3000 such lines in the packet-switching network; a total of about 7000 lines. A hypercube network with the same number of processors would require about 10 000 such lines.
- Since the network in CONNECT can be built out of many independent modules, such as a subtree, a small hypercube and the Clos network, the cost should be much less than that of building a complete network all together.
- CONNECT requires a number of additional communication processors, which is a cost not present in networks with active switching components.

Combining the pros and cons discussed above, our feeling is that the cost of CONNECT should be comparable to, or less than, a single network of a comparable complexity.

CONCLUSIONS

CONNECT has shown the potential of building large scale parallel architectures based on limited hardware building blocks. It is a hybrid circuit-switching and packet-switching network and provides advantages that the two types of networks cannot provide individually. The details of the circuit-switching network and the packet-switching network can still have many variations and be radically changed, but the philosophy of a hybrid network is probably a solution to many architectures.

As an example of an application, our ongoing research on a database machine, called the Linear-throughput Semantic Database Machine (LSDM)[25, 28–30], uses CONNECT as its hardware base. In LSDM, each data processor of CONNECT is equipped with a disc drive, providing massively parallel I/O capacity. The combination of highly parallel computing power and parallel I/O power makes LSDM a machine of an extremely high throughput.

Applications of CONNECT in other areas are yet to be explored. Since CONNECT tends to have a large storage space and high computation power, its applications in knowledge-based systems and AI systems should be examined.

Some possibilities for improvement remain, which the authors intend to investigate in the future. For example, due to the limitations imposed by the building blocks, the control of the circuit-switching network is semi-distributed, instead of fully distributed.

Due to the nature of the Clos network, it is difficult to expand the network dynamically. The expansion needs either adding more stages or using switching components of higher connectivity. This problem will be studied further.

The software mechanism which deals with the component failure dynamically is yet to be developed and the performance under a partial system failure is to be studied.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the advice of David Barton, Doron Tal, Nagarajan Prabhakaran, Samuel Shapiro, Neil Wiseman, Eduardo Fernandez and Scott Graham. Our special thanks go to the anonymous referees for their valuable and detailed comments and suggestions, which have resulted in a significant improvement of the paper.

REFERENCES

1 **Hillis, W D** *The Connection Machine* MIT Press, USA (1985)
2 **Lipovski, G J and Tripathi, A** 'A reconfigurable varistructured array processor' in *Proc. 1977 International Conference on Parallel Processing,* (1977) pp 165–174
3 **Lipovski, G J and Malek, M** *Parallel Computing Theory and Comparison* John Wiley (1987)
4 **Gajski, D D, Lawrie, D H, Kuck, D J and Sameh, A H** 'CEDAR' in **Hwang, K (Ed)** *Supercomputing – Design and Applications,* IEEE (1984) pp 251–275
5 **Gottlieb, A *et al.*** 'The NYU ultracomputer-designing an MIMD shared-memory parallel computer' *IEEE Trans. Comp.* Vol C-32 No 2 (1983) pp 175–189
6 **Batcher, K E** 'Design of a massively parallel processor' *IEEE Trans. Comp.* Vol C-29 No 9 (1980)
7 **Siegel, H J, Schwederski, T S, Davis, N J and Kuehn, J T** 'PASM: a reconfigurable parallel system for image processing' *ACM SIGARCH Newsletter* Vol 12 No 4 (1984) pp 7–19
8 **Chow, E, Maden, H and Peterson, J** *Hyperswitch Network for Hypercube Concurrent Computer,* Technical Report, Jet Propulsion Laboratory (1986)
9 **Chow, E, Maden, H, Peterson, J, Grunwald, D and Reed, D** 'Hyperswitch network for the hypercube computer' in *Proceedings of the 15th International Symposium on Computer Architecture* (June 1988) pp 90–99
10 **Seitz, C L** 'The cosmic cube' *Comm. ACM.* Vol 28 No 1 (January 1985) pp 22–23
11 **Snyder, L** 'Introduction to the configurable, highly parallel computer' *Computer* (January 1982) pp 47–56
12 **Shaw, D E** 'The NON-VON supercomputer' (1982)
13 **Pfister, G *et al.*** 'The IBM research parallel prototype (RP3): introduction and architecture' in *Proceedings of the 1985 International Conference on Parallel Processing* (August 1985) pp 764–771
14 **Rudolph, J A** 'A production implementation of an associative array processor STARAN' in *Proc. of the Fall Joint Computer Conference* Las Vegas, Nev (November 1972) pp 229–241
15 **Hwang, K and Briggs, F** *Computer Architecture and Parallel Processing* McGraw-Hill (1984)
16 **Leiserson, C** 'Fat-trees: universal networks for hardware-efficient supercomputing' *IEEE Trans. Comp.* Vol C-34 No 10 (1985) pp 892–901
17 **Dally, W J** 'A VLSI architecture for concurrent data structures' *PhD thesis,* California Institute of Technology (1986)
18 *Transputer Architecture Reference Manual* Inmos Corporation, Bristol, UK (1986)
19 **Stone, H S** *High-Performance Computer Architecture* Addison-Wesley (1987)
20 **Wu, C L and Feng, T Y** *Tutorial: Interconnection Networks for Parallel and Distributed Processing* IEEE Computer Society Press (1984)
21 **Clos, C** 'A study of nonblocking switching networks' *Bell Syst. Techn. J.* (March 1953) pp 406–424
22 **Benes, V E** 'On rearrangeable three-stage connecting networks' *Bell Syst. Techn. J.* (August 1962) pp 1481–1492
23 **Rishe, N and Li, Q** 'A proof of impossibility of deadlock in a fixed-routing hypercube network' in *Proc. of the Fourth Conference on Hypercube Concurrent Computers* Monterey, CA (March 1989) in press
24 **Dally, W J and Seitz, C** 'Deadlock-free message routing in multiprocessor interconnection networks' *IEEE Trans. Comp.* Vol C-36 No 5 (May 1987) pp 547–553
25 **Li, Q** 'The architecture and related control problems of a transputer based highly parallel database machine' *PhD thesis* Florida International University, Miami (December 1989)
26 **Li, Q, Wang, L and Rishe, N** 'A three processor grouping problem in the hypercube networks' *Technical Report SCS TR 89-07* Florida International University, Miami (August 1989)
27 **Despain, A M and Patterson, D A** 'X-tree: a tree

Qiang Li received a BE degree in electrical engineering from the Xi'an Jiaotong University, Xi'an, China, in 1982, and MS and PhD degrees from Florida International University, Miami, FL, USA in 1984 and 1989, respectively. He is currently an Assistant Professor of Computer Engineering at Santa Clara University, Santa Clara, CA. His research interests include parallel processing, parallel architectures, operating systems, and simulation.

Dr Rishe is an Associate Professor of Computer Science at Florida International University. Dr Rishe's publications on databases and related issues include two books (Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology, *Prentice-Hall, 1988;* Database Design: The Semantic Modeling Approach, *McGraw-Hill, 1992) and 50 papers. Dr Rishe chaired the steering and programme committees of the PARBASE-90 conference and is on the steering committee of the PDIS conference series. Dr Rishe also has extensive experience in data base applications and data base systems in industry. This included eight years of employment as head of software and database projects (1976–84) and later consulting for companies such as Hewlett-Packard. Prof. Rishe has a PhD in computer science from Tel Aviv University, Israel.*

structured multiprocessor computer architecture' in *Proceedings of the Fifth Annual Computer Architecture Symposium* (1978) pp 144–151

28 **Rishe, N, Tal, D and Li, Q** 'Architecture for a massively parallel database machine' *Microprocessing Microprog.* Vol 25 (1988) pp 33–38

29 **Rishe, N** *Database Design Fundamentals: A Structured Introduction to Databases and a Structured Database Design Methodology* Prentice-Hall, Englewood Cliffs, NJ (1988) pp 1–40

30 **Rishe, N** 'Efficient organization of semantic database' in **Litwin, W and Schek, H-J (Eds)** *Foundation of Data Organization and Algorithms* Springer-Verlag (1989) pp 114–127