MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 130

Packet Communication Architecture

by

Jack B. Dennis

(A paper to be published in the Proceedings of the 1975 Sagamore Computer Conference on Parallel Computation)

This research was supported in part by the National Science Foundation under grant DCR75-04060 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0661.

August 1975

PACKET COMMUNICATION ARCHITECTURE

Jack B. Dennis
Project MAC
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract -- Packet Communication Architecture is the structuring of data processing systems as collections of physical units that communicate only by sending information packets of fixed size, using an asynchronous protocol. Each unit is designed so it never has to wait for a response to a packet it has transmitted to another unit while other packets are waiting for its attention. Packets are routed between sections of a system by networks of units arranged to sort many packets concurrently according to their destination. In this way, it is possible to arrange that system units are heavily used provided concurrency in the task to be performed can be exploited. The packet communication principle is especially attractive for data flow processors since the execution of data flow programs readily separates into many independent computational events. In this paper we show how packet communication can be used in the architecture of memory systems capable of processing many independent memory transactions concurrently and having hierarchical structure. The behavior of these memory systems is prescribed by a formal memory model appropriate to a computer system for data flow programs.

Introduction

With the advent of LSI technology, the main direction of further advance in the power of large computer systems is through exploitation of parallelism. Attempts to achieve parallelism in array processors, associative processors and vector or pipeline machines have succeeded only with the sacrifice of programmability. These large parallel machines all require that high levels of local parallelism be expressed in program formats that retain the notion of sequential control flow. Since most algorithms do not naturally exhibit local parallelism in the form expected by these machines, intricate data representations and convoluted algorithms must be designed if the potential of the machine is to be approached.

The alternative is to design machines that can exploit the global parallelism in programs, that is, to take advantage of opportunities to execute unrelated parts of a program concurrently. Conventional sequential machine languages are unsuited to this end because identification of concurrently executable program parts is a task of great difficulty. Data flow program representation are of more interest, for only essential sequencing relationships among computational events are indicated. An instruction in a data flow program is enabled for execution by the arrival of its operand values -- there is no separate notion of control flow, and where there is no data dependence between program parts, the parts are implicitly available for parallel execution.

Several designs for data processing systems have been developed that can achieve highly parallel operation by exploiting the global concurrency of programs represented in data flow form [1-6]. Two of these designs [3,6] are able to execute programs expressed in a conventional high-level language that exceeds Algol 60 in generality. These systems consist of units that operate independently and interact only by transmitting information packets over channels that connect pairs of units. The units themselves may have similar structure so the system as a whole has a hierarchical structure that we call packet communication architecture.

In this paper we discuss the principles of packet communication architecture, and illustrate their application to the organization of memory systems for highly concurrent operation.

The Packet Communication Principle

Suppose the data processing part P of the computer in Figure 1 is organized so many independent computational activities may be carried forward concurrently -- as would be true if P contains many independent sequential processors, or if P is designed to exploit the inherent parallelism of data flow programs. Activities in P will generate many independent requests to the memory system M for storage or retrieval of information. It is not essential that M respond immediately to these requests because, if P is properly organized, its resources (registers, instruction decoders, functional units) may be applied to other activities while some activities are held up by pending memory transactions. Thus the memory system need not be designed to complete one transaction before beginning the processing of other transactions. In fact, we will see how this freedom can be exploited in memory systems organized to process many transactions concurrently and keep their constituent units heavily utilized.

This is the first principle of packet communication architecture -- designing each part of a system so many activities are concurrent, and reasonable delays in information transmission between parts may be incurred without important loss of performance. This tolerance of delay permits a radically different approach to the design of switching mechanisms that interconnect sections of a computer system.

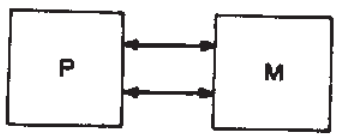


Figure 1. Computer system.

Figure 2. Application of a routing network.

In a system designed to profit from this principle, the response to a request may arrive only after many additional requests have been sent. In general the responses will occur in a different order than the requests raising the problem of how the requesting system part can relate each response to the request that generated it. In many cases it is possible to avoid the problem by including sufficient information in the response that the processing of it is determined without relating it to a specific request. This principle is followed in the several proposed data flow architectures and leads to some elegant data processing structures.

A function frequently required in a computer system is a mechanism to direct requests from one system part to the appropriate specialized unit of the system according to the nature of the request. Two examples are: an instruction and its operands must be sent to an appropriate functional unit for execution; a request by a processing unit for the contents of a specified memory location must be sent to the appropriate memory module. Figure 2 illustrates the former example in the case of a data flow processor [1]. Operands arrive at units called Instruction Cells to form Operation Packets which must be transmitted to the appropriate Functional Units. Some form of switching mechanism must provide a path for Operation Packets between each Instruction Cell and each Functional Unit. Because attaining minimum delay is a less crucial objective in a packet communication system than achieving high concurrency, Routing Networks such as shown in Figure 3 are an attractive form of switching mechanism.

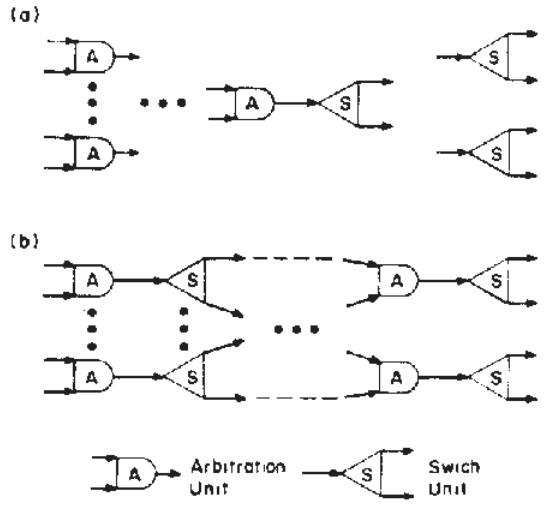


Figure 3. Routing network structure.

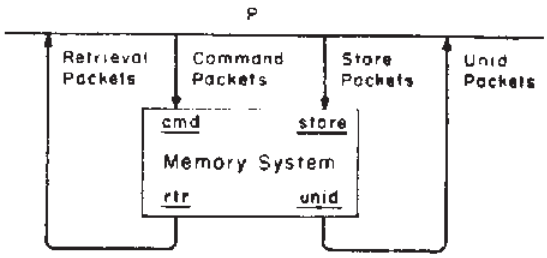


Figure 4. A memory system and its ports.

A routing network is itself a packet communication system built of two basic kinds of modules: Arbitration Units and Switch Units. An Arbitration Unit transmits packets arriving on either input channel to its output channel. A Switch Unit sends arriving packets over the output channel determined by some property of the packet.

If the Arbitration Units and Switch Units are grouped as in Figure 3a, there is a single channel through which all packets must flow -- probably a bottleneck. Alternatively, interleaving ranks of Arbitration Units and Switch Units as indicated in Figure 3b provides for concurrent flow of packets over disjoint paths.

A routing network that has few input ports, and therefore consists mostly of Switch Units is called a Distribution Network, and has the effect of sorting arriving packets according to their contents. In Figure 2 the routing network would sort operation packets according to the operation codes of the instructions they contain. A routing network that has few output ports, and consists mostly of Arbitration Units is called an Arbitration Network.

Packet Communication Memory Systems

As an example of packet communication architecture, consider the memory system shown in Figure 4 which is connected to a processing system P by four channels. Command Packets sent to the memory system at port cmd are requests for memory transactions, and specify the kind of transaction to be performed. Items to be stored are presented as Store Packets at port store, and items retrieved from storage are delivered as Retrieval Packets at port rtr. The role of port unid will be explained later.

For further discussion of the operation of this memory system, we must define the desired behavior -- the nature of the information stored, and how the contents of Retrieval Packets depends on the contents of Store Packets previously sent to the memory system. A precise specification of behavior may take the form of an abstract memory model consisting of a domain of values and a specification of each transaction in terms of the sequences of packets passing the ports of the memory system. We give an informal outline of such a memory model.

For simplicity, the value domain V is

$$V = E + [V \times V]$$

and is the union of pairs consisting of all ordered pairs of elements of V. This domain is recursively defined, and consists of all finite binary trees having elementary values at their leaves.

Our memory model must deal with the retention of information by the memory system. We use a domain of memory states which are acyclic directed graphs called state graphs. Each node of a state graph represents a value (binary tree) in V in the obvious way.

The transactions of this memory model are so specified that no outgoing arc is added or deleted from a node already present in the state graph, and hence the value represented by a node never changes. A memory system having this property is attractive for applicative languages such as pure Lisp and various determinate data flow languages. However, such a memory model is incomplete in that it does not support the updating of a shared data base, for example. The proper way to generalize this memory model is a matter of current research.

The basis of a memory state is a subset of the nodes of a state graph that includes every root node of the graph (Thus each node and arc of a state graph is accessible over a directed path from some basis node). Each basis node represents a value in terms of which the processing system may request transactions by the memory system.

Each node of a state graph has an associated reference count which is the sum of two numbers--the number of state graph arcs that terminate on the node, and the number of "references" to the node (if it is a basis node) held in the processing system P. Each node of a valid state graph must have a reference count greater than zero.

We regard the memory system as holding a collection of items that represent a state graph in the manner of a linked list structure. To this end we require a set of unique identifiers for the nodes of state graphs. One may regard each unique identifier as corresponding to a unique site in the memory system that can hold a distinct item. The items held by the memory system are of two kinds:

1. Elementary items: ⟨elem, i, e, r⟩
   where i is a unique identifier
   e is an elementary value
   r is a reference count
2. Pair Items: ⟨pair, i, j, k, r⟩
   where i, j, k are unique identifiers
   r is a reference count

Elementary items and pair items correspond to leaf nodes and pair nodes, respectively, of a state graph. In each item, i is the unique identifier of the item.

For the purpose of specifying the transactions of the memory system, it is convenient to suppose that it has the structure shown in Figure 5. Command Packets delivered at port rc (for reference count) of M are merged with Command Packets from P and presented to M at port cmd. We specify the behavior of the whole memory system by specifying the behavior of M. We regard the state of M as consisting of a collection of items and a collection of unique identifiers not in use. In the initial state of M the collection of items is empty and every unique identifier is not in use.

The specifications for the behavior of M state the response, if any, and change of state,

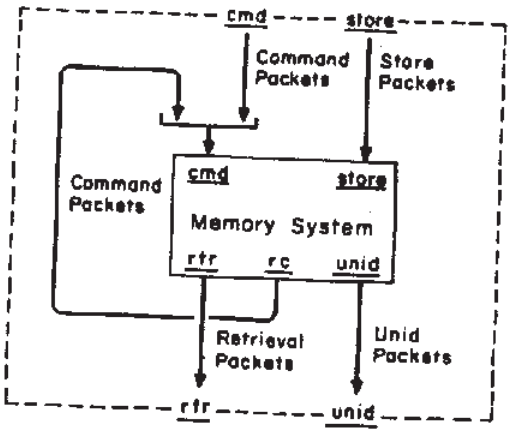


Figure 5. Structure of a memory system for specification of its transactions.

if any, that accompany each kind of transaction. In the simple memory system we are considering, there are five kinds of transactions -- four of these are associated with acceptance of Command Packets by M, and the fifth is associated with delivery of Unid Packets. The behavior of M for each kind of transaction is as follows.

Store Transaction:

⟨store, i⟩ → cmd

{⟨elem, i, e⟩ ⟨pair, i, j, k⟩} → store

In response to a store Command Packet, the item presented at port store is added to the collection of items held by M, and is given an initial reference count of one.

Retrieval Transaction:

⟨retr, i⟩ → cmd

rtr → {⟨elem, i, e⟩ ⟨pair, i, j, k⟩}

The item dilivered at port rtr is the item with unique identifier i in the collection of items held by M. The state of M does not change.

Reference Generation and Anihilation

(a)⟨up i⟩ → cmd

(b)⟨down i⟩ → cmd

(c)⟨down i⟩ → cmd

rc → ⟨down i⟩ ⟨down k⟩

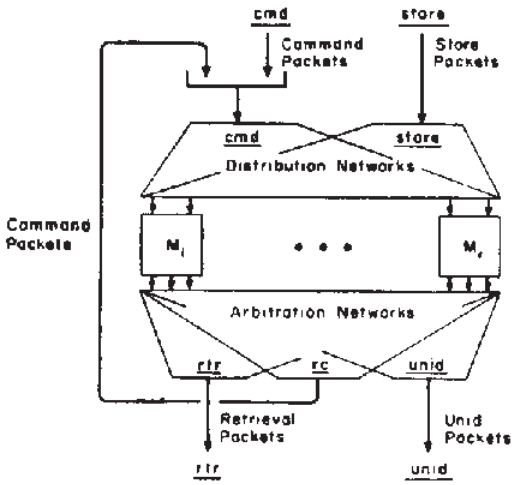


Figure 6. Memory system structure for concurrency.

The up command adds one to the reference count of item i; the down command decrements its reference count by one. If the reference count is reduced to zero by a down command, the item is deleted from the collection of items held by M and its unique identifier i is added to the collection of unused unique identifiers. Case (c) applies if the item deleted is a pair item since the reference counts of its component items must be decremented.

Unique Identifier Generation:

Some unique identifier i is removed from the set of unused unique identifiers and delivered at port unid.

We have not specified the behavior of M under certain conditions that should not occur during normal operation -- for example, if a store Command Packet contains a unique identifier which is already the unique identifier of an item held by M. We assume the processing system is so designed that such ill behavior cannot occur. A discussion of these restrictions on processor behavior and how they might be implemented is beyond the scope of the present paper.

Memory System Structure

With this albeit informal specification of M, we are prepared to see how M may be realized by simpler subsystems interconnected as a packet communication system.

First we show how concurrent processing of many transactions can be achieved by distributing

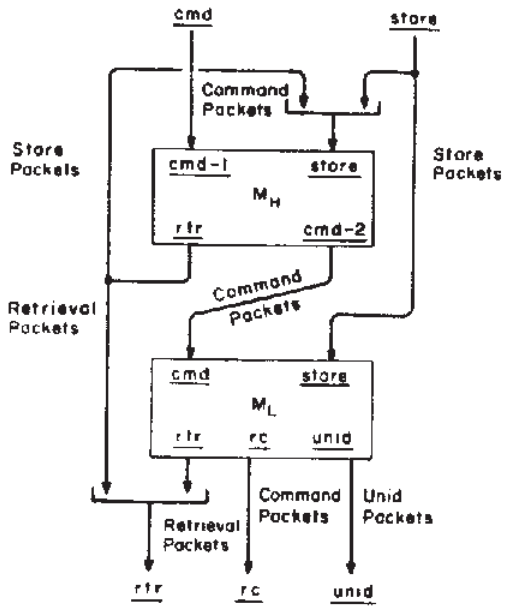


Figure 7. Hierarchical packet memory system.

Command Packets among many identical physical modules which can operate independently. Such a structure for M is shown in Figure 6. Each Command Packet and each Store Packet is distributed to one of the memory subsystems $M_1 \ldots M_r$ according to some easily tested property of i, the unique identifier of the item to which the packet refers. The property might be the first p bits of the binary representation of the unique identifier where $r = 2^p$.

The subsystems $M_1, \ldots, M_r$ are memory systems having specifications identical to the specification of M except that the universe of unique identifiers for the items held by each subsystem is restricted to $(1/2)^p$ of the unique identifiers of M. This fact may be used to reduce the complexity of the memory subsystems.

The Retrieval Packets, Command Packets, and Unid Packets delivered by the memory subsystems at their rtr, rc, and unid ports are merged into common streams by three Arbitration Networks. Note that the Command Packets from subsystem rc ports must be recirculated through the Distribution Networks because, in general, the items they refer to will be held in subsystems other than the subsystem from which they originate.

Hierarchical Structure

Any realistic design for a large memory system must recognize the principle that only the most active information need be held in expensive fast-access devices; less active information should be held in slower devices. If the memory system is to support modularity of programming in its most general form, then information must be automatically redistributed among levels of the memory system as computational activity involves different portions of the stored information.

Figure 7 shows how a memory system M satisfying our specification may be realized by a hierarchical organization of two memory systems M_H and M_L. With an important exception explained later, the lower level subsystem M_L satisfies the same behavioral specification as the entire memory system M. The higher level subsystem M_H is arranged to hold copies of the most active items in M_L -- it acts as a cache memory so M is able to achieve a much lower latency in processing transactions than M_L could alone. Keep in mind that even though M_L may have a long latency, it may have a high rate of processing transactions due to its organization for highly parallel operation.

If there is no room in M_H for an item sent to M for storage, or for an item retrieved form M_L, then some item is selected for deletion from M_H. The criterion for selecting the item to be deleted could be any of the schemes used in contemporary cache memories or paging systems. The deleted item need not be sent to M_L because the memory system under discussion is organized so M_L holds a copy of every item present in the memory system M. However, M_L must know which items it holds have duplicates in M_H so it can tell whether it is safe to release the unique identifiers of deleted items for reuse. Hence each item in M_L includes an indicator f that tells whether the item is also held in M_H:

Elementary Items: ⟨elem, i,e,r,f⟩
Pair Items: ⟨pair, i,j,k,r,f⟩

where f is one of {true, false}

The tansactions of M_L have specifications just like those for the transactions of M, except for a few changes:

Store Tansactions: The indicator f of the item added to the collection is true since each Store Packet is sent to both M_H and M_L.

Anihilation: If the reference count of an item is reduced to zero, the item is deleted and its unique identifier released for reuse only if the indicator f is false.

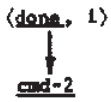
Done Transaction: An additional form of Command Packet ⟨done, i⟩ is sent by M_H to M_L to say that item i has been deleted from M_H. Subsystem M_L responds by setting f to false and, if the reference count is zero, the item is deleted from M_L and its unique identifier is released for reuse.

In M_H items are held without reference counts:

Elementary Items: ⟨elem, i,e⟩
Pair Items: ⟨pair, i,j,k⟩

For the purpose of specifying the behavior of M_H, its state is simply a collection of items in these formats. A realization of M_H would require additional state information to implement the chosen criterion for deletion of items. Item removal is a routine used by store transactions of M_H:

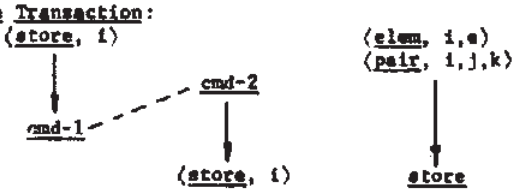
Item Removal:



The item to be removed is deleted from the collection of items held by M_H and a done Command Packet is sent at port cmd-2.

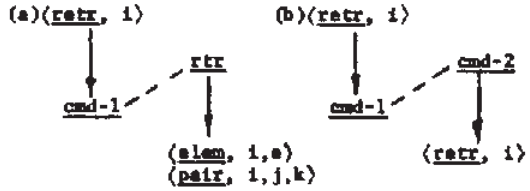
The transactions of M_H are:

Store Transaction:



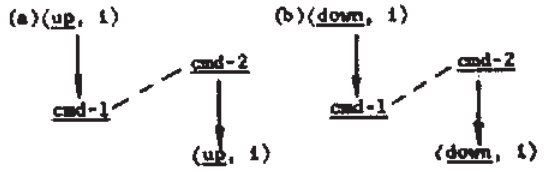
The item is added to the collection after some chosen item is removed, if necessary, to create space. The store Command Packet is forwarded to M_L.

Retrieval Transaction.



Case (a) applies if an item with unique identifier i is in the collection held by M_H; otherwise case (b) applies, and the retrieval Command Packet is forwarded to M_L.

Reference Accounting:



Up and down Command Packets are forwarded to M_L without action by M_H.

In addition, for correct operation of the whole memory system M, all Command Packets relating to item i must be delivered at port cmd-2 in the same order as they arrive at port cmd-1.

Conclusion

Packet Communication architecture offers many attractions to the computer system designer: The units of a system interact through very simple interfaces and are easy to specify; timing hazards are eliminated through use of a strict speed independent communication discipline; and the principles are applicable to the organization of machines that support the execution of well structured programs expressed in high level languages. A high level of concurrent operation can be achieved with high equipment utilization if the global parallelism inherent in most data processing tasks is expressed in the program.

It will be interesting to see if these attractions can be realized in the design of practical computer systems.

Acknowledgment

This research was supported in part by the National Science Foundation under grant DCR75-04060 and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0661.

Bibliography

1. Dennis, J. B., and D. P. Misunas, "A computer architecture for highly parallel signal processing," Proceedings of the ACM 1974 National Conference, ACM, New York (November, 1974), 402-409.

2. Dennis, J. B., and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," Proceedings of the Second Annual Symposium on Computer Architecture, IEEE, New York (January 1975), 126-132.

3. Misunas, D. P. A Computer Architecture for Data-Flow Computation. SM Thesis, Department of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass. (June 1975).

4. Misunas, D. P., "Structure processing in a data-flow computer," Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, IEEE, New York (August 1975).

5. Project MAC Progress Report XI, Project MAC, M.I.T. (July 1973-74), pp. 84-90.

6. Rumbaugh, J. E., A Parallel Asynchronous Computer Architecture for Data Flow Programs, Project MAC, M.I.T., Cambridge, Mass., Report TR-150 (May 1975).