MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Laboratory for Computer Science

Computation Structures Group Memo 128-1

# Performance Analysis of a Data-Flow Processor

by

David P. Misunas

(A paper to be published in the Proceedings of the 1976 Sagamore Computer Conference on Parallel Computation)

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract N00014-75-C-0661.

September 1976

# PERFORMANCE ANALYSIS OF A DATA-FLOW PROCESSOR*

David P. Misunas
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract -- A data-flow processor is structured as a packet communication system. Sections of a processor are connected by interconnection networks which have a great deal of inherent parallelism, and the sections communicate by means of fixed size information packets. The processing capability of a data-flow processor is determined through consideration of the flow of packets within the interconnection networks, and the actual performance of the processor is affected by the structure of the networks. The execution time of an instruction in a processor can vary greatly due to conflict within the interconnection networks. The performance of a data-flow processor is measured through consideration of the delays caused by this conflict, and the proper network structure and processing rate of a machine are determined through analysis of the best and worst case delays.

## Introduction

Efforts to develop a model of computation which can effectively express parallelism have yielded a new form of program representation known as data flow [1,2,3,6,7,8,10]. The attractiveness of data flow lies in the fact that it is data-driven: that is, an instruction is enabled for execution only after each required operand has been provided by the execution of a predecessor instruction.

We have been conducting architectural studies to investigate the design of a processor which can efficiently execute data-flow programs by taking advantage of the parallelism inherent in the data-flow representation. The resulting architectures [4, 5] offer attractive solutions to some of the problems of parallel systems. The usual problems of processor switching and memory/processor interconnection are avoided by the use of interconnection networks which have a great deal of inherent parallelism. The structure of the processor allows a large number of instructions to be active simultaneously. These active instructions pass through the networks concurrently and form streams of instructions for the pipelined functional units.

*This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-C-0661.

Initial investigations culminated in the development of an architecture for a processor that executed programs expressed in the elementary data-flow language [4]. The elementary language incorporates no fancy capabilities such as recursion, data structures, conditionals, or iteration. However, the language and its corresponding architecture are well-suited for the representation and execution of signal processing computations such as filtering, waveform generation, fast Fourier transforms, and so forth.

The next step involved developing the architecture of the basic processor [5]. This machine and its corresponding language incorporate conditional and iterative mechanisms and a multi-level memory system in which the active memory is operated as a cache, and individual instructions are retrieved from the auxiliary memory as they become required for computation.

The most recently developed machine in this series expands the architecture and language to incorporate procedures, recursive activation, and data structures represented as acyclic directed graphs [8, 9]. A more conventional approach to the implementation of a complete data-flow language has been developed by Rumbaugh [11, 12].

The performance of a data-flow processor is analyzed through consideration of the flow of information within the interconnection networks of the processor. In illustration of this technique of performance analysis, we consider such an analysis of the performance of an elementary data-flow processor.

## The Elementary Data-Flow Processor

The computational capability of the elementary data-flow processor is limited to programs expressed in the elementary data-flow language. A program in this language is constructed of two kinds of elements, called operators and links. Operators are represented as circles with a number of input arcs and one output arc. A link is designated by a small dot and receives results from an operator on its input arc and distributes them to other operators over its output arcs.

Tokens are represented by large solid dots and convey values over the arcs of the program. An

operator with a token on each of its input arcs and no token on its output arc is enabled and sometime later will fire, removing the tokens from its input arcs, computing a result using the values associated with the input tokens, and associating that result with a token placed on its output arc. Similarly, a link is enabled when a token is present on its input arc and no token is present on any of its output arcs. It fires by removing the token from its input arc and associating copies of the value carried by the input token with tokens placed on its output arcs.

In Figure 1 we have a rather simple data-flow program. There is a value present on each input arc, and thus links L1 and L2 are enabled. Either one can fire -- suppose L1 does. Then operator A2, which multiplies its input by the constant A, and link L2 are enabled. Once again, either A2 or L2 can fire, and in this manner tokens travel through the program until a token appears on the output conveying the value Ax(x+y). Once operators A1 and A2 have fired, there are no tokens on the arcs emanating from L1 and L2, and the links can fire as soon as two new input values arrive. Thus, these elementary programs can readily represent pipelined computation.

The Memory of the elementary data-flow processor shown in Figure 2 holds a representation of the program to be executed. This Memory is a collection of Instruction Cells (Figure 3); one Instruction Cell is associated with each operator of the program. Each Instruction Cell is composed of three registers, the first of which specifies the operation to be performed and the address(es) of the register(s) to which the result of the operation is

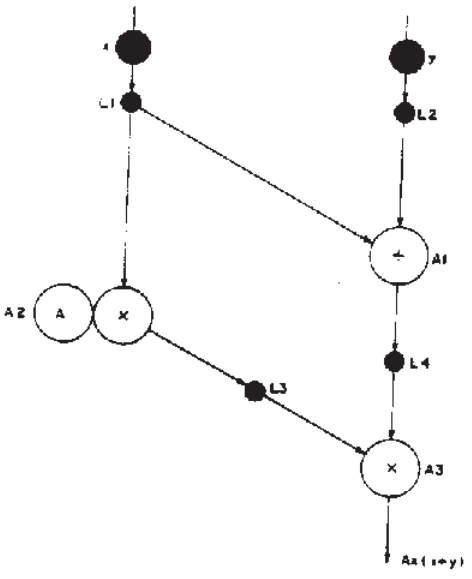



Figure 1. An elementary data-flow program.

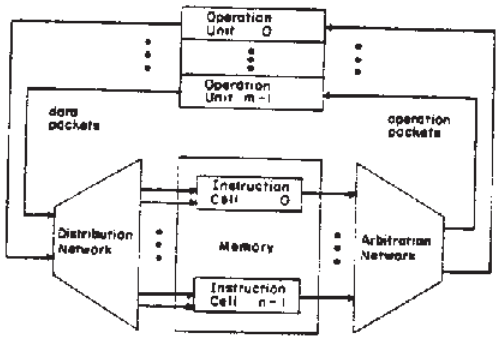



Figure 2. Structure of the elementary data-flow processor.

to be directed. The second and third registers receive operands for use in execution of the instruction.

When an Instruction Cell contains an instruction and all required operands, the Cell is said to be enabled and presents its contents as an operation packet to the Arbitration Network for delivery to an Operation Unit which can perform the desired function. The Arbitration Network provides a path from each Instruction Cell to each Operation Unit. The network is capable of simultaneously accepting many operation packets from the Instruction Cells and delivers each packet to an appropriate Operation Unit by decoding the instruction portion of the packet.

Upon receiving an operation packet, an Operation Unit performs the function specified by the instruction on the operands of the packet and produces a data packet, containing one copy of the result and a destination register address, for each destination specified in the instruction. A Distribution Network concurrently accepts data packets from the Operation Units and, using the destination address of each packet, delivers it to the specified register of the Memory. The

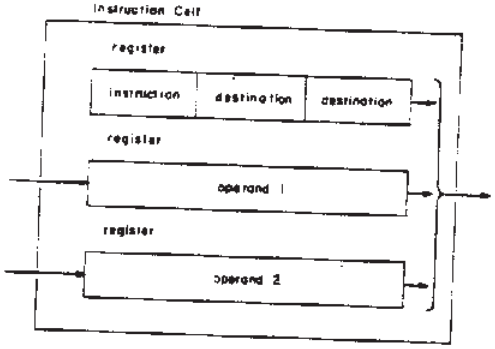



Figure 3. Structure of an Instruction Cell

Instruction Cell containing that register may then be enabled if an instruction and all operands are present in the Cell.

A simplified structure of the Arbitration and Distribution Networks is presented in Figure 4. The networks are composed of three types of units. An arbitration unit passes packets arriving at its input ports one-at-a-time to its output port, using a round-robin discipline to resolve any conflicts. A switch unit passes a packet at its input to one of its outputs, controlled by some property of the packet. In the Arbitration Network this property is the operation code, whereas in the Distribution Network, the switch units are controlled by the destination address. A buffer unit stores a packet until the succeeding switch or arbitration unit is ready to accept it.

Due to the large number of inputs to the Arbitration Network, we wish to transfer data between the Memory Cells and the Arbitration Network in serial format to reduce the number of wires necessary. However, in order to maintain a high rate of packet flow at the output ports, we wish to transfer packets to the Operation Units in parallel format. For this reason, serial-to-parallel conversion is done gradually within the buffer units as a packet travels through the Arbitration Network. Parallel-to-serial conversion is performed in the Distribution Network for similar reasons.

## Processor Performance

To analyze the performance of the elementary data-flow architecture, we must consider the utilization of the Instruction Cells of the Memory; that is, the number of times a Cell will be enabled within a given time period. This will then allow us to determine the processing rate of the machine.

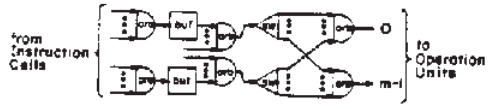
The execution cycle time of an instruction within the processor is the minimum elapsed time between the enabling of the instruction and the arrival of the result of the operation specified by the instruction at the desired destination Cell(s). For an instruction of the elementary data-flow processor, the execution cycle time is equal to the passage time through the Arbitration Network, the Distribution Network, and an appropriate Operation Unit. The delay in the Operation Unit is fixed for that Operation Unit. However, the network delays can vary greatly due to the presence of conflict.

The execution cycle time for an instruction is found by considering the passage of the operation packet containing that instruction through the Arbitration Network and the passage of the resulting data packets through the Distribution Network with no conflict. The minimum delay through a network, the Arbitration Network for example, is given by the summation over the number of stages in the network of the time required to transfer a packet through each stage:

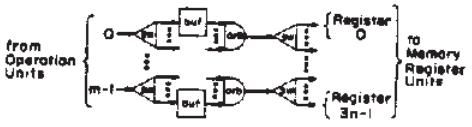
$$\sum_{\text{stages}} (\text{no. bits serial} + 1)(\text{bit transfer time})$$

The transfer time for a stage is equal to the number of bits passing through the stage in serial plus one for a signal to indicate that the packet is ready to be transferred multiplied by the time necessary to transfer a bit. A similar equation applies to delay in the Distribution Network.

Let us examine the delay within a specific Arbitration Network (Figure 5). This network has three stages and seven arbitration units. Packets travel through stage 0 in four-bit serial format and are gradually converted to a more parallel format, passing through stage 1 in two-bit serial and stage 2 in one-bit serial format. As noted previously, the passage time for a packet through each stage is equal to the number of serial bits plus one times the bit transfer time t. For the structure of Figure 5, the transfer times are 5t, 3t, and 2t, respectively. The minimum delay through the network is equal to the summation of the stage delays, or 10t.




(a) Arbitration Network




(b) Distribution Network

Figure 4. Structure of the Arbitration and Distribution Networks.

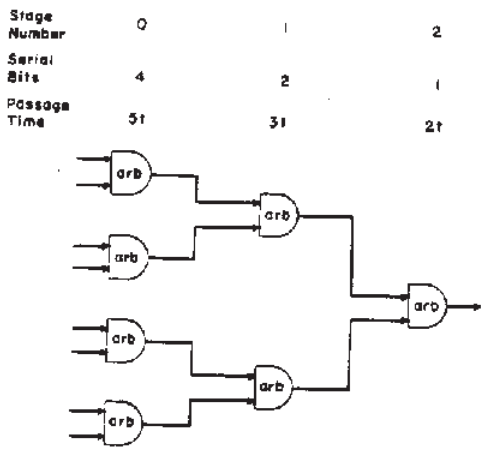



Figure 5. Structure of an elementary Arbitration Network.

To find the time T necessary to process all instructions contained in the Memory of the processor, we must consider the maximum delay a packet can encounter in passing through the Arbitration Network. Such a maximum delay can occur in a network which has a packet present at every node in a machine in which every Instruction Cell is enabled, placing a packet on each input to the Arbitration Network (Figure 6). The maximum delay which can be encountered by a packet, say the triangular one, arises only when all other packets in the network pass through the output of the network before the triangular one does. In order for this to happen, not only must the triangular packet lose every conflict, but every packet on the path it will follow to the output must also lose every conflict. Thus, finding the maximum delay involves determining how many packets will flow through each stage before the triangular one.

For this network, the worst case packet will be the 14th through stage 2, the 6th through stage 1, and the 2nd through stage 0. Multiplying the number of packets passing through each stage by the delay in that stage, we find that:

T = maximum delay
= 2(3t) + 6(3t) + 14(2t)
= 56t

Hence, if all instructions of the processor are enabled, they can pass through this Arbitration Network in a maximum time of 56t.

However, if we assume that the network size is such that the execution cycle time is less than T, then a number of destination Cells become enabled and enter the Arbitration Network before all Cells have been processed, and the processing rate of the machine can be measured in terms of the output rate of the Arbitration Network (assuming the Distribution Network has been structured to distribute all results as fast as they are produced). In such a case, the rate of packet transfer to each Operation Unit is 1/(2t), and the maximum processing rate of the machine is [1/(2t)](number of Operation Units).

Furthermore, if each arbitration unit has enough inputs to allow a packet to travel through the previous stage in less time than that required to service all busy inputs, the passage of the triangular packet through the first stages of the Arbitration Network will occur simultaneously with the transmission of other packets at the output of the network. The time T for the transmission of all packets in the network to the Operation Units is then 14(2t) = 28t.

## Network Structure

The results developed in the previous section seem to indicate that a network of as few stages as possible is desirable in order to decrease the execution cycle time and increase the number of inputs to an arbitration unit of the network. In general, this is true. However, the fact that packets are transferred from each Instruction Cell in serial format requires a number of stages in the Arbitration Network in order to perform the conversion to parallel format before a packet

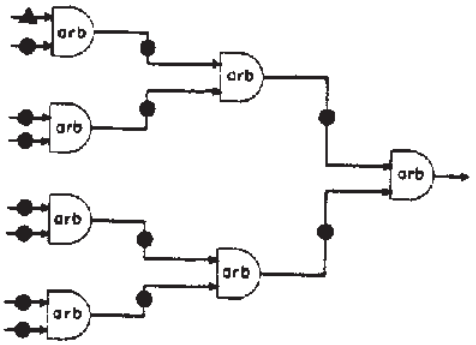



Figure 6. Example of a full Arbitration Network.

reaches the final stage of arbitration. Also, a number of stages are necessary in order to maintain a queue of instructions for each Operation Unit.

The actual structure of the Arbitration Network does not significantly affect performance as long as a few simple rules are observed in its construction. If $D_{Ai}$ is the passage delay of a packet through stage i of the Arbitration Network, and $I_{Ai}$ is the number of inputs to stage i, then the following relationship must hold:

$$D_{Ai} = \alpha[(I_{A(i+1)})(D_{A(i+1)})], \quad \alpha<1$$

This assures that each stage of the Arbitration Network is kept busy by the preceding stages.

The value of the constant $\alpha$ is dependent upon the utilization of the machine. Since the processor is designed to support pipelined computation, the value of $\alpha$ is controlled by the amount of the machine which is used for computation and the difference between the sample input rate and the maximum processing rate.

The addition of a switch unit at the output of an arbitration unit introduces a further factor for consideration. If $S_{Ai}$ is the number of outputs of the switch unit after stage i of arbitration, then

$$D_{Ai} = \alpha[(I_{A(i+1)})(D_{A(i+1)})]/S_{Ai}$$

and the number of inputs to the arbitration units of stage i+1 must be increased by the number of outputs of the switch unit of stage i in order to keep the arbitration unit in stage i+1 busy.

Similarly, the Distribution Network must be structured so that

$$D_i = \alpha[(S_i)(D_{(i+1)})]/I_i$$

where $S_i$ is the number of outputs of the switch unit in stage i, $I_i$ is the number of inputs of the arbitration unit preceding the switch unit of stage i, and $D_i$ is the delay through stage i of the network.

## An Example Processor

An illustration of the capability of an elementary data-flow processor, consider the execution of a highly parallel, pipelined computation on a 128 Instruction Cell machine in which all Cells are fully utilized. The Instruction Cells of the example machine accept and transmit packets in 16-bit parallel, 4-bit serial format.

For a balanced processor structure, one in which the number of Operation Units is matched to the number of Instruction Cells, the processing time T should be equal to the minimum delay D through the networks and an Operation Unit. Thus, to determine the optimal number of Operation Units for the processor, we must consider the structure of the networks in order to discover the minimum delay.

To obtain a small execution cycle time, and hence, a greater processing capability, the networks must be structured with as few stages as possible. However, three stages are required in the Arbitration Network to perform the serial-to-parallel conversion and still maintain the necessary throughput from stage to stage. The minimum delay analysis of this three stage network structure is identical to that described in the previous section; the delay in the Arbitration Network is equal to 10t.

Assuming that the minimum delay in the Distribution Network and the delay in an Operation Unit are the same as that in the Arbitration Network, the resulting value for D is:

$$D = 30t$$

If t = 150 nanoseconds, allowing 15 TTL gate delays to accomplish one ready/acknowledge cycle, the resulting execution cycle time is :

$$D = 30(150 \text{ nsec.})$$
$$= 4.5 \text{ microseconds}$$

To establish the number of Operation Units necessary for a balanced processor structure, with a stage delay of 300 nsec. for each pipelined Operation Unit, we must set the processing time T for all enabled instructions contained in the Memory equal to the execution cycle time:

$$T = 4.5 \text{ microseconds}$$
$$= (128)(300 \text{ nsec.})/(\text{no. of Operation Units})$$

yielding:

$$\text{No. of Operation Units} = 9$$

And the resulting performance of the processor is:

$$\text{Processing rate} = 128 \text{ instructions} / 4.5 \text{ microsec.}$$
$$= 28 \text{ MIPS}$$

## Conclusion

There are a number of ways in which the processing rate of a data-flow processor can be extended. First, the size of the Instruction Memory and the number of Operation Units can be increased. If the additional Cells are fully utilized, the processing rate will grow linearly with the number of Cells added. Second, the bottlenecks of the machine, the output of the Arbitration Network and the input of the Distribution Network could be fabricated in a faster technology. A change from TTL to ECL at the bottlenecks should allow a five-fold increase in the processing rate. Naturally, the slower portions of the networks must be structured in more parallel forms to maintain this rate. A technology change would also allow a decrease in the number of Operation Units if they were to be constructed of the faster technology.

## References


[1] Adams, D. A., A Computation Model With Data Flow Sequencing, School of Humanities and Sciences, Stanford University, Stanford, Calif., (December, 1968).

[2] Bährs, A., "Operation Patterns (An Extensible Model of an Extensible Language)," Symposium on Theoretical Programming, Novosibirsk, USSR, (August, 1972).

[3] Dennis, J. B., "First Version of a Data Flow Procedure Language," Lecture Notes in Computer Science 19, (G. Goos and J. Hartmanis, Eds.), Springer-Verlag, New York (1974), pp. 362-376.

[4] 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), pp. 402-409.

[5] 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), pp. 126-132.

[6] Karp, R. M., and R. E. Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing," SIAM Journal of Applied Mathematics 14 (November, 1966), pp. 1390-1411.

[7] Kosinski, P. R., "A Data Flow Language for Operating Systems Programming," Proceedings of the ACM SIGPLAN-SIGOPS Interface Meeting, SIGPLAN Notices 8, (September, 1973), pp. 89-94.

[8] 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).

[9] 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), pp. 230-234.

[10] Rodriguez, J. E., A Graph Model for Parallel Computation, Report TR-64, Project MAC, M.I.T., Cambridge, Mass., (September, 1969).

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

[12] Rumbaugh, J. E., "A Data Flow Multiprocessor," Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, IEEE, New York (August, 1975), pp. 220-223.