MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE

Computation Structures Group Memo 174

Data Flow Computer Architecture

(Research Proposal Submitted to the National Science Foundation)

March 1979

CONTENTS

1. Introduction

We propose to continue our program of basic research on data flow computation -- program structure and computer architecture -- currently funded by the National Science Foundation under grant MCS75-04060 A01.

Data flow research at MIT has been funded by NSF for basic studies of implementation issues arising from our architectural proposals, and by the Lawrence Livermore Laboratory (calendar 1978) for evaluating the potential of our approach to data flow computation for energy-related high performance computations. For the next phase of our research program, we anticipate receiving funding from the Department of Energy to support construction and evaluation of data flow computers according to our architectural proposals. We wish to continue our work on the conceptual and theoretical foundations of data flow computation and packet communication architecture with NSF support.

Understanding of the principles of data flow computer organization and user language design has reached the stage that we are confident that practical computer systems of this kind can be built. In fact, interest in the subject has spread both in this country and abroad [Gur77, Pat78, Ti79, Tre77], and several experimental machines have been built [Dav78,Syr77, Ti79].

At the MIT Laboratory for Computer Science we are engaged in a long-term research program for developing the basic concepts of data flow program organization and execution, and applying them in the design and evaluation of programming languages and proposals of computer architecture for data driven computation. The ultimate goal of this work is to specify computer systems that are capable of high performance at low cost by exploiting advanced logic and memory technologies, and that support sound principles of program structure and language design. To achieve this goal it is essential that the conception of computer architecture and the design of the programming language to be supported go hand in hand. Thus our architectural proposals are *language-based* in the sense that a machine built according to our proposal will be capable of correctly and effectively executing any program written in a well-specified base language so long as resource limitations are not exceeded. While language-based design is important in any innovative computer system, it is crucial for data flow machines because they are sufficiently different from von Neuman machines that conventional software methodologies simply do not apply.

2. The MIT Data Flow Architecture

The architectural concepts under study in the MIT Computation Structures Group are based on use of packet transmission between hardware units and use of routing networks to communicate information between sections of the machine.

Our research has led to a series of architectural proposals, each designed to support a specific level of programming language expressive power. The principles are best introduced in terms of the most basic structure, shown in Fig. 1, which we call our Form I data flow processor.

The Form I machine has four sections connected by packet transmission channels;

Memory Section -- consists of Instruction Cells which hold instructions and their operands.

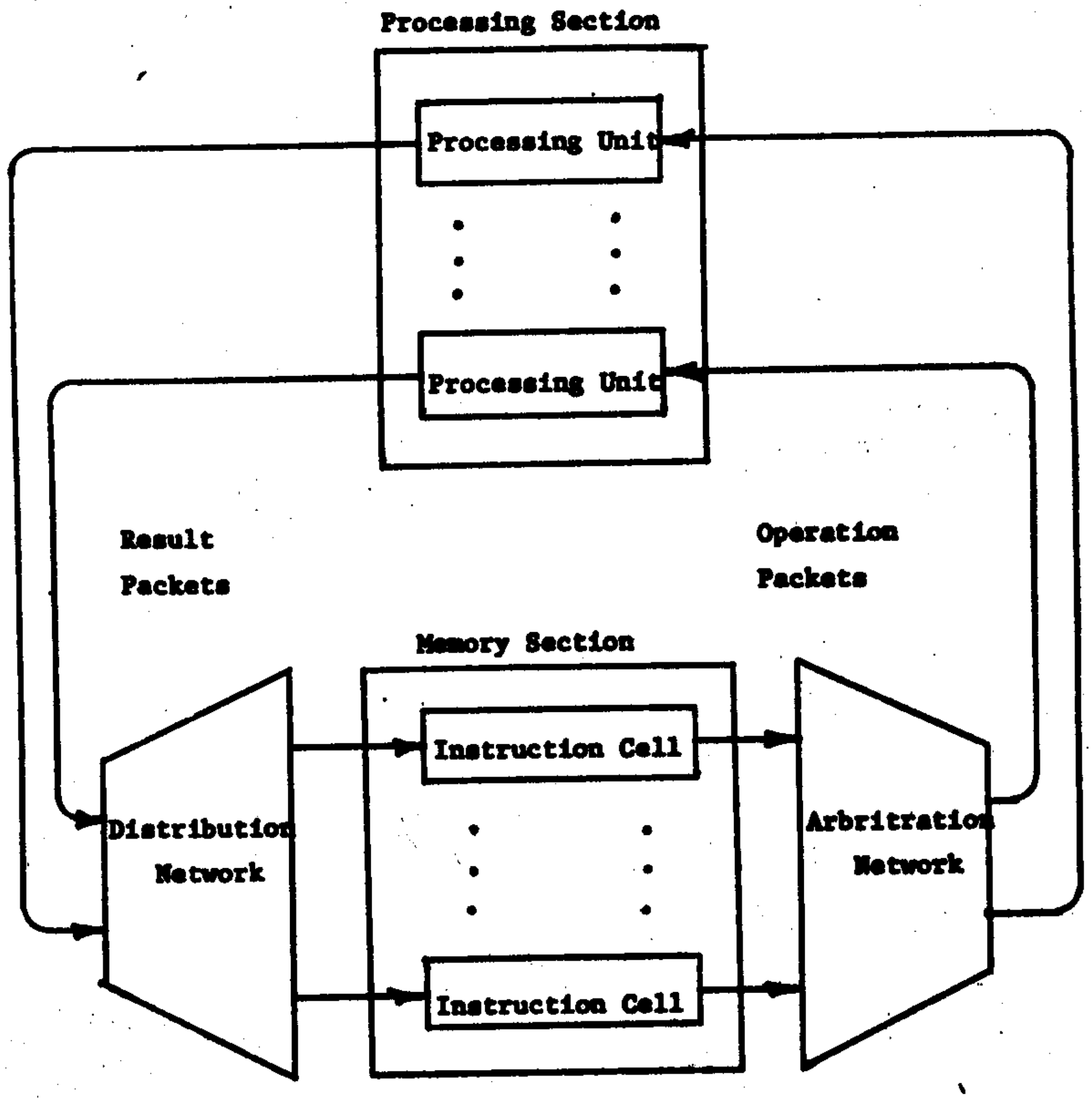
Processing Section -- consists of Processing Units that perform the basic scalar operations on data values.

Arbitration Network -- delivers Operation Packets from the Memory Section to the Processing Section.

Distribution Network -- delivers Result Packets from the Processing Section to the Memory Section.

The overall operation of these sections is best summarized in terms of their packet communication. Instructions held in the Memory Section are enabled for execution by the arrival of their operands in Result Packets from the Distribution Network. Enabled instructions, together with their operands, are sent as Operation Packets to the Processing Section through the Arbitration Network. The results of instruction execution are sent

Fig. 1. Form 1 data flow processor.



through the Distribution Network to the Memory Section where they become operands of other instructions. A more complete description of the operation of these sections can be found in [Den77-1].

Form 1. The Form 1 processor corresponds to a basic language level supporting scalar variables, and having conditional and iteration control structures. Since all data and instructions reside in the Instruction Cells, this form of data flow machine is suitable for fast computations involving relatively small programs and data. This is the form of data flow machine that has been studied most intensively at MIT. A comprehensive account of its structure, operation, and application to the fast Fourier transform can be found in [Den77-1]. This machine seems well suited to a wide variety of numerical computations including many signal processing applications.

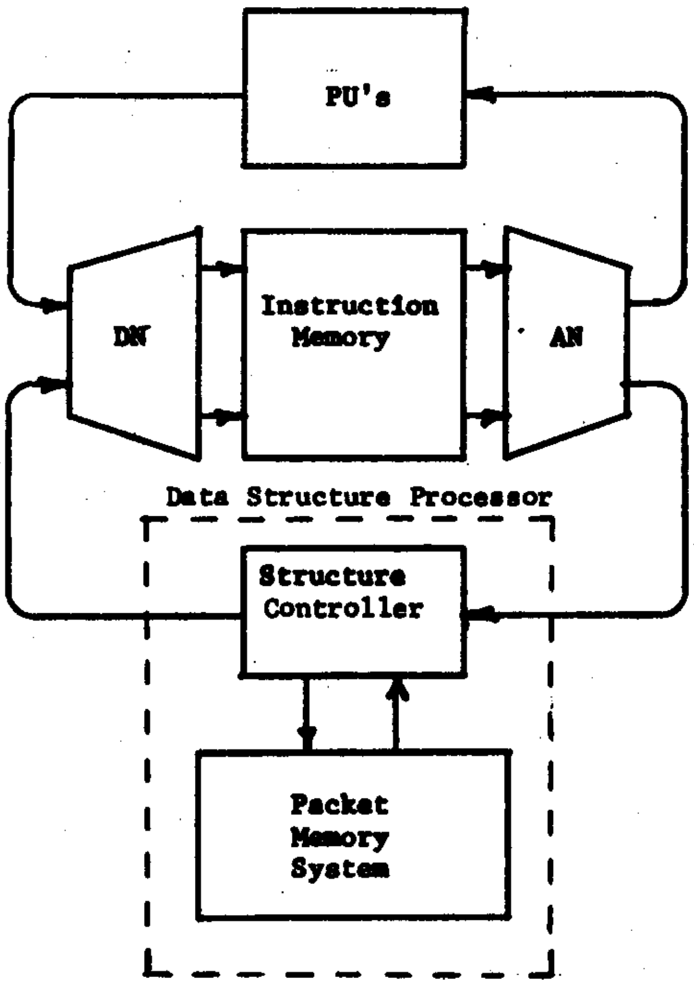
Form 2. A Form 2 data flow machine is obtained by adding to the Form I machine a Data Structure Processor consisting of a Structure Controller and a Packet Memory System, as shown in Fig. 2. Correspondingly, the language supported is extended to include a general class of data structures and operations for their construction and access. Since a program is still held in the Instruction Cells, the program size limitation of the Form 1 machine still applies. However, the Structure Processor may be designed to handle very large data bases.

A general discussion of Form 2 data flow machines prepared for the Symposium on High Speed Computer and Algorithm Organization [Den77-2] outlined its application to global weather simulation, a problem requiring high performance and a large data base.

Form 3. A Form 3 data flow processor supports the same language level as a Form 2 machine, but allows the execution of large programs. This is accomplished by implementing the Instruction Memory (as a packet memory system), as shown in Fig. 3, and arranging that only the most active instructions are held in Instruction Cells during execution of a program. Thus the Instruction Cells act as a "cache" for the Instruction Memory.

The basic machinery needed to make the Instruction Cells act as a cache has been presented in [Den75-1] as a generalization of the basic Form I architecture. This work must be revised, extended, and evaluated in the context of more recent developments, in particular deadlock prevention and schemes for implementing data structures.

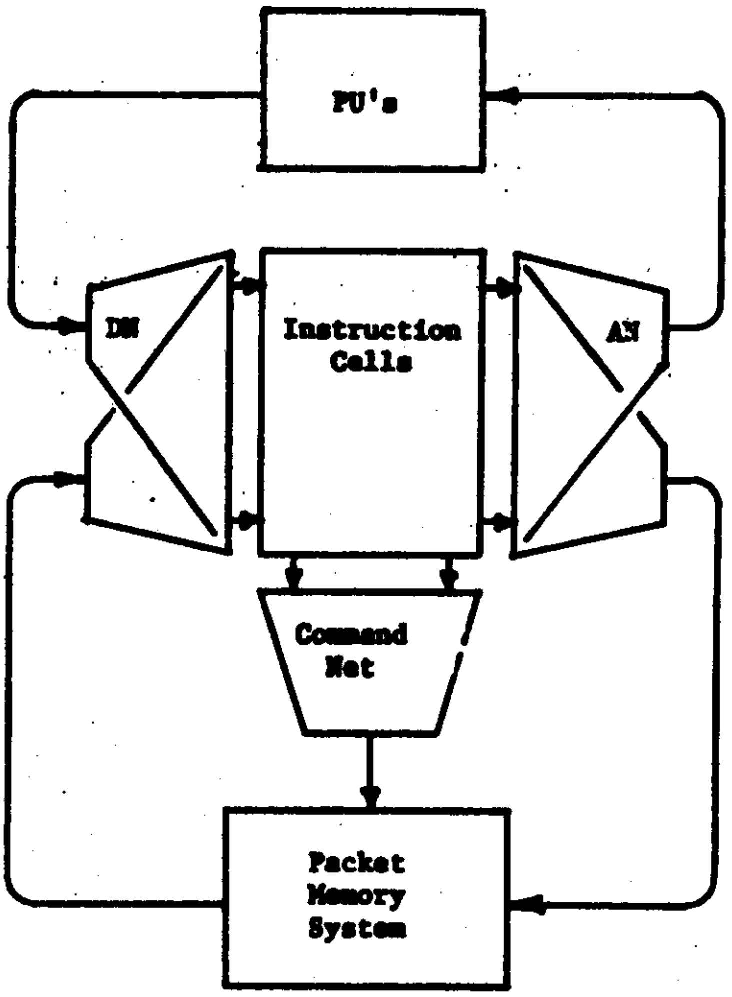
Fig. 2. Form 2 data flow machine.



Form 4. A Form 4 data flow computer is envisioned as supporting, at the hardware level, all fundamental aspects of data driven computations, including procedures, recursion, and data streams. These machines would be sufficiently general to support all services of a general purpose computer system for a community of users.

Solving the conceptual and design problems of a Form 4 data flow computer is a very ambitious task. The major problem holding up progress toward a complete specification of a Form 4 computer is the design of a procedure execution mechanism that will operate effectively in the most general context. Contributions toward a solution to these problems have been made by Misunas [Mis78] and Miranker [Mir77] at MIT, and work at Irvine

Fig. 3. Form 3 data flow processor.



[Arv79] and Utah [Pat78] provides a rich source of ideas. Currently, Weng is studying this problem as his doctoral research at MIT.

These proposed architectures differ from the proposals of other research groups in several significant ways:

1. We propose routing networks as the basic means of communicating operation and result packets. These structures have logic complexity N log N and delay log N which is as good as can be done while maintaining throughput in proportion to N.

2. Communication between two instructions has the same delay and expense regardless of where the two instructions are located. Thus the performance realized in an application does not depend on partitioning the program into regions of high locality. Consequently, code generation is more straightforward.

3. Our structure processor concept is further developed than other proposals for implementing data structures in a manner consistent with data flow principles.

3. Research Status

Research progress in the past two years may be divided into language design and translation; data flow semantic theory; data flow computer architecture; principles of packet architecture; and studies of application areas.

3.1 Language Design and Translation

With support from the Livermore Laboratory, a major effort during the past year has been the design of the programming language VAL (Value-oriented Algorithmic Language). The purpose of developing VAL is to provide a medium for expressing applications in a form suitable for evaluating their suitability for data flow computation. Our goals in the design of VAL have been to allow expression of concurrency, to support good program structure, to suit the computational physics applications of interest to Livermore, and to be insofar as practical a "general purpose" language.

Existing languages suitable for computational physics applications reflect the storage structure of the von Neuman machine in that each language provides some means of effecting a change in memory which can not be modeled as a local effect. Such languages permit programs to be written which are very difficult or impossible to analyze for parts that may be executed concurrently.

In contrast, VAL is a *functional* or *applicative* language: each module or well formed portion of a VAL program corresponds to a mathematical function and the entire effect of putting two parts together is to compose the corresponding functions. We have given careful consideration to the recently developed body of knowledge about program structures and language characteristics which support program verification. We have found a natural consistency between language design for support of concurrency and language design for correctness and verifiability. This has made it possible in the design of VAL to adhere to program structures and language characteristics that have been found desirable for ease of understanding and verification, and ease of building a program by combining separately specified modules.

Innovative features of VAL include: an iteration construct based on representing iteration in the form of tail recursion; a **forall** construct that permits simultaneous computation of the elements of a new array; and value-oriented treatment of exceptions through inclusion of a special error element in each data type.

While other value-oriented programming languages have been developed (pure LISP [McC60] and LUCID [Ash77] are earlier examples), we believe VAL is the first which is seriously intended for writing large scale programs for efficient numerical computation on high performance machines. A preliminary reference manual for VAL has been written and will be published as a Technical Report in March 1979 [Ack79].

Translating programs written in VAL into efficient data flow machine language requires translation and optimization techniques significantly different from those used with conventional languages and machines. Translation from a value-oriented language such as VAL into a data flow graph is straightforward due to the absence of side effects. On the other hand, machine language programs for our Form 1 and Form 2 architectures are not guaranteed to be "safe". By "safe" we mean that an Instruction Cell never receives a result packet when the Instruction Cell is not prepared to accept it. Machine programs that are safe can be obtained by having each instruction send *acknowledge packets* to predecessor instructions as illustrated in [Den77-1]. Rules for inserting acknowledge signalling to achieve high concurrency without changing the effect of executing a data flow program have been developed by Montz [Mon79].

3.2 Semantics

The objective of our work in semantics is to develop a sound mathematical theory for the semantics of data flow computation. In a recent work Brock presents two semantic specifications for a simple, determinate subset of VAL and proves their equivalence. The operational semantics of a program in this subset is obtained by a two-step process. First, a translation function maps the program into a data flow graph. Rules for this translation are given in the thesis. Next, an interpreting function maps the data flow graph into the result of executing the data flow graph. Because determinate programs may be translated into determinate data flow graphs, the fixed point methods of Kahn [Kah74] may be used to determine the result of graph execution. The denotational semantics of a program is defined using Scott's theory [Sco76] by directly mapping program language elements into a mathematical domain. The proof of equivalence for the two semantic methods is also a "proof of correctness" for the translation and interpreting functions. The formal development of the operational semantics is presented in [Bro78-1].

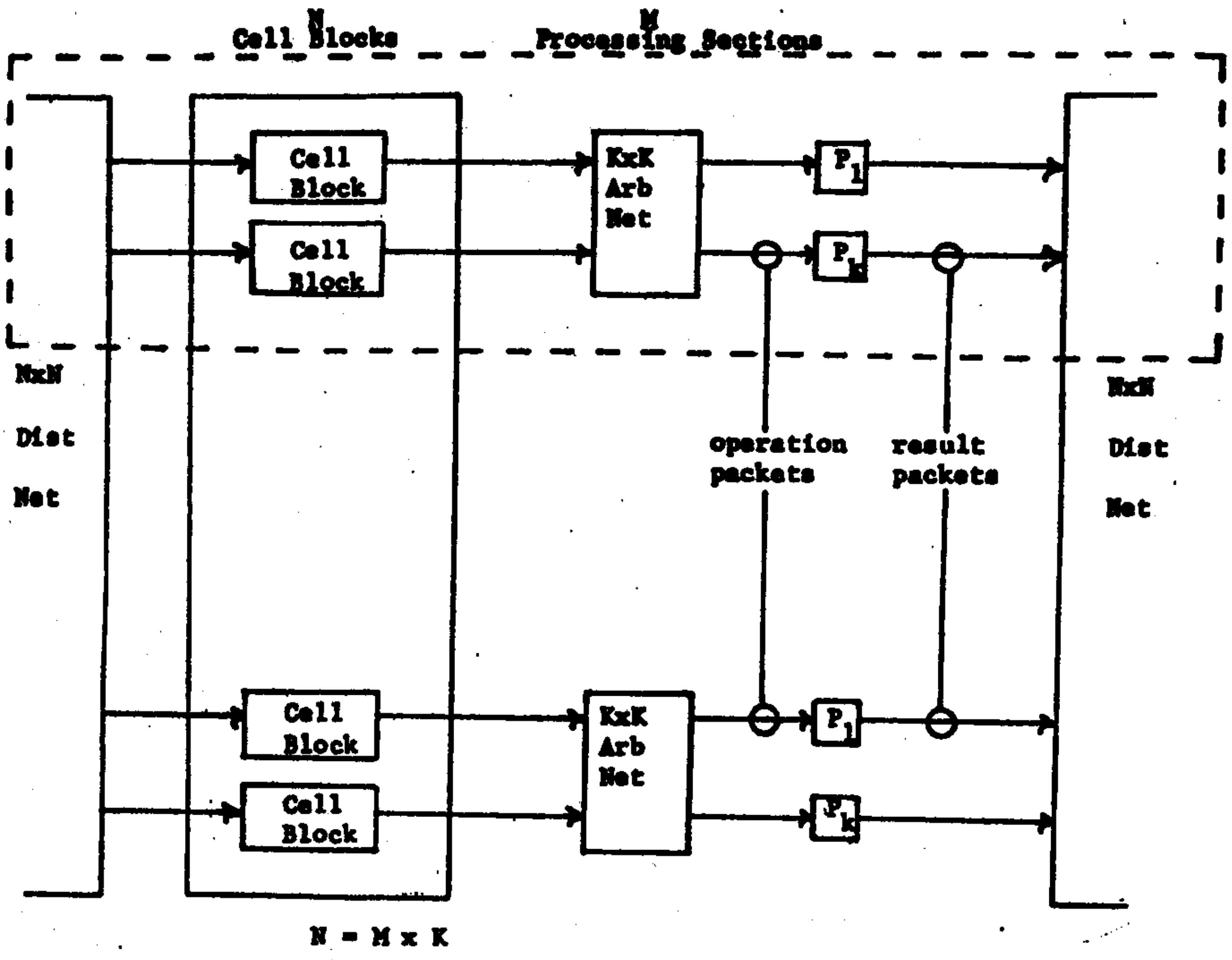
3.3 Data Flow Computer Architecture

During the next two years we plan to build an experimental Form 1 data flow processor with support from the Department of Energy. The objective is to gain experience with the technical issues of implementing systems using packet architecture principles, to solve the problems of translating real programs for efficient data flow computation, and to gain understanding with the development and debugging of application programs.

The envisioned form of the experimental processor is shown in Fig. 4. In contrast with our earlier reports, each information packet will be represented as a sequence of eight-bit bytes throughout the processor. Instruction Cells are grouped into Cell Blocks, each containing sixteen or more Instruction Cells. There are many Processing Units of K different types which process packets in byte serial format.

In preparation for this project we have conducted design studies for the three basic units of the architecture: the routing networks, the Cell Block and the Processing Units.

Fig. 4. M.I.T. proposed engineering model.



Routing Networks

A routing network is a packet communication system designed so each packet arriving at an input port is eventually routed to the output port corresponding to a tag field of the packet. An arbitrary routing network can be built using two simpler types of networks called concentration and connection networks. A concentration network has more inputs than outputs, and has the property that each accepted packet is eventually placed on an output, but is not necessarily routed to a particular output. A connection network is a routing network with the same number of inputs as outputs. In [Bou78] Boughton has developed efficient concentration network designs, and has shown that connection networks which have probabilistically high throughput can be constructed from concentration networks and switches.

A concrete design of an Arbitration Network has been developed in a recent bachelor's thesis [McN78].

Instruction Cell Block

A specification for the Instruction Cell Block has been prepared and the structure of an implementation has been developed as a thesis project by Amikura [Ami77]. Deciding on the technology and scale of integration, and completing detailed logic design for the experimental Form 1 processor remains to be done.

Processing Units

The design of arithmetic processors suitable for on-line byte-serial operation has been studied by Feridun [Fer78]. The use of signed-digit number representation allows use of addition algorithms in which carries are propagated at most one digit position, and serial data may be processed most significant digit first [Avi61,Avi64]. Similar algorithms exist for multiplication and division [Tri77]. Based on these algorithms, architectures for a floating-point adder-subtractor and a floating-point multiplier have been developed.

We expect to extend our experimental Form 1 processor to be a Form 2 machine by adding a Structure Processor. The basic ideas have been presented in [Den75-2], and a specific design has been explored by Ackerman [Ack77]. The Structure Processor consists of a Packet Memory System that holds representations of data structure values (records and arrays), and a Structure Controller that interprets high level data structure operations as commands to the packet memory. The data structure operations implemented by the Structure Controller are value-oriented, as required for consistency with data flow semantics. This requires copying of portions of data structures when modified structure values are generated by a program. In Ackerman's design of the Structure Processor, care has been taken to minimize such copying while allowing concurrency of data structure operations to be exploited. The Structure Processor is designed to have high performance by handling large numbers of structure operations concurrently, and a modular organization is proposed such that the system has no bottlenecks and the processing rate may be increased indefinitely by adding modules.

3.4 Packet Architecture

Packet system architecture is a new context in which to study the various aspects of computer system description, correctness, performance and reliability. Our research includes contributions in each of these areas which will support our program to construct machines using the principles of packet architecture.

Architecture Description Language

In constructing a practical data flow processor, it is invaluable to have a formal description of its architecture. An architecture description should specify the major subsystems of an architecture, their logical connection in terms of the information passed among them, and the processing each performs on this information. An architecture description is essentially an operational model for the semantics of machine language programs written for the data flow processor. We have identified a set of concepts suitable for this task: modules, module ports for input/output, data flow semantics for algorithmic behavior specification, PASCAL-type records for data structures, module state variables for processing packet streams, and monitors for resource sharing. These concepts have been incorporated into a proposal for an architectural description language (ADL) that is explained and illustrated in [Leu79-1]. In terms of this ADL, the internal structure of an architectural unit can be specified as an interconnection of simpler units, and its external behavior can be expressed in terms of receiving, processing, and generating information packets. Hierarchical architectural specifications can thus be constructed. Early versions of our ADL have been illustrated in [Leu75, Den77-1].

Correctness and Verification

We have begun a serious effort to develop a methodology for specifying and verifying hardware systems that is analogous to recent developments in software methodology. In hardware systems, concurrency is an essential factor, and normal operation often continues indefinitely without termination; an appropriate methodology must reflect these differences from usual program characteristics. In his doctoral thesis, David Ellis has developed a formal basis for specifying and verifying packet systems [Ell77]. As in our ADL, this theory views a packet system both *behaviorally* in terms of its interaction with its outside world, and *structurally* as a composition of smaller systems. Ellis shows how packet systems may be proved correct by establishing that the formal characterization corresponding to these two views are equivalent.

Fault Tolerance

In his doctoral thesis research [Leu79-2], Clement Leung is studying the problems of designing a fault tolerant data flow processor. The data flow processor is assumed to have a packet system architecture and to be constructed from self-timed modules which communicate via asynchronous packet transmission protocols. A survey of classical fault tolerance techniques and methods for constructing self-timed modules has been completed. This study leads to a proposed hardware structure for packet modules. An *elementary* packet module contains registers for receiving inputs and a combinational circuit for packet processing. A *compound* packet module is an interconnection (cyclic or acyclic) of elementary packet modules. Due to the use of asynchronous protocols, the appropriate fault model for packet modules turns out to be random pulse trains on wires, as opposed to, say, the classical stuck-at fault models. This fault model has been used to characterize the symptoms of hardware failures at the packet communication level and provides a basis for studying fault tolerance techniques.

3.5 Application Studies

We have found many computational problems requiring high performance to be well suited for data flow execution. The Fast Fourier transform and a global weather model have been mentioned as attractive applications for Form 1 and Form 2 data flow computers. More recently a hydrodynamics code representative of applications at the Livermore Laboratory has been reexpressed in a preliminary version of VAL and analysis of its potential for data flow execution is being studied. In addition, students in an MIT graduate subject have completed 15 term projects evaluating data flow computation for problems ranging from differential file comparison to radar signal analysis.

4. Proposed Research

With National Science Foundation support we plan to continue our basic studies, concentrating in four areas: program optimization and application throughput analysis; principles of packet system architecture; semantic theory for data flow programs; and implementation schemes for data structures, procedures and streams.

4.1 Program Optimization and Analysis

Efficient program execution on data flow computers requires a good match between the structure of programs and the structure of the machines on which they are to be run. Clearly, to fully utilize the capability of a data flow computer for concurrent computation, the application problem must offer much opportunity for parallelism. In our studies of potential application areas for high performance data flow computation we have found no shortage of parallelism: Signal processing computations are described by flow diagrams for which the computation required in each block may proceed concurrently with the others. The important fast Fourier transform for N samples can be computed in log N time using N-fold parallelism. The problems of computational physics often amount to independent calculations at each node of a large two- or three-dimensional grid. The global weather simulation used at the Goddard Institute for Space Science uses a grid of 72 x 45 x 9 points. For each time step, independent computation is done for each point of the grid.

In fact, the problem in many applications is to *reduce* the parallelism to a degree commensurate with the capabilities of the data flow processor. In many cases the most natural expression of a computation is one that expresses it in fully parallel form. For example

```
old, new: array [array [real]]
new := forall i in (1, m), j in (1, n)
           construct f(old, i, j)
```

defines a rectangular matrix new in terms of a similar matrix using the computation expressed by f. This computation could be carried out on a Form 1 data flow machine by loading m x n copies of the machine level program for f into Instruction Cells, thereby realizing the m x n-fold parallelism of the problem. For large computations (e.g., the weather problem) this would require ridiculously large numbers of Instruction Cells.

Using a Form 2 data flow computer, an alternative implementation is to store the matrix old in the data structure memory. The matrix would be sent as a stream of element values by the structure processor to Instruction Cells holding one copy of the machine level program for f. The stream of results computed by f would be returned to the structure processor to be retained as elements of the matrix new. The concurrency exploited in this version is the pipelined operation of the fetch, compute, and store parts of the task and the parallelism represented in the implementation of f.

An intermediate possibility is to organize the computation so the m rows of data are presented in succession by Instruction Cells holding n copies of the machine level program for f.

For a large-scale computation many choices of machine level program structure are possible, and there is a large space of options in which to find the best match of problem to machine. We propose to study program transformations, such as the array-to-stream conversion discussed above, that would make possible automatic or user-guided generation of optimum program structure for a given application on a specified data flow computer.

The nature of program optimization for data flow computation is an interesting contrast to optimization for conventional computers: for conventional machines it is the inner loops and detailed structure of the machine level program that matters (subject to consideration of program size); in data flow computation, it is the outer levels of program organization that have the most significant influence on overall performance. In view of this contrast, new concepts and principles are needed to guide the generation of good programs for Form 1 and Form 2 data flow machines intended for high performance numerical computation.

The rate of computation for a data flow program on a Form 1 data flow processor is governed by cycles in the program graph representation of the machine program. If the transmission times for operation and result packets through the routing networks may be assumed constant, the theory of timed-Petri-nets [Ram73] may be used to determine the computation rate of a periodic data flow computation. Computation rates should provide useful information for balancing instruction cell allocation to different program parts to achieve the best throughput for the entire program. The details remain to be worked out.

4.2 Packet System Architecture

We have found packet communication architecture to be a very attractive form in which to realize a high performance data flow computer. Large data flow computers built according to our architectural concepts will consist of hundreds or thousands of units communicating by means of packet transmission. Since asynchronous computers of this form and complexity have not been designed previously, we plan to continue our basic studies related to the specification and realization of packet communication systems.

Architecture Description Language

We have already identified a set of concepts suitable for describing packet systems at the architectural level. The next step is to synthesize these concepts into an architecture description language (ADL). ADL should contain language constructs to support the aforementioned concepts and be properly human engineered.

A formal architecture description is just a starting point in constructing a packet system. The challenge is to devise a practical machine that is provably equivalent to the one specified in an architecture description. An attractive approach to meet this challenge is to successively refine descriptions in ADL to the point where a description can be systematically translated into hardware. The choice of additional refinement techniques for ADL and techniques for translating from high level descriptions into asynchronous LSI hardware structures are both topics for research. In the course of this study we also expect to benefit from the experience to be gained in constructing a prototype Form 1 data flow processor. Properties of an ADL description can be verified mathematically, or by interpretation. We plan to program an ADL interpreter which can be extended to be a packet system simulator and used to both verify designs and evaluate the performance of data flow computer architectures.

Hardware Specification and Verification Techniques

Specification and verification techniques play important roles in a structured hardware design methodology for constructing provably correct packet systems. Consider an implementation of a packet module M by an interconnection of submodules $M_1, ..., M_n$. Given the external behavior of the M_i's, we need to verify that interaction among them is indeed consistent with the external behavior desired for M. To achieve mathematical rigor in the verification process, the external behavior of M, $M_1, ..., M_n$ and their interaction must be formally specified. Verification of implementation correctness then requires exhibiting a mathematical proof of equivalence between the external behavior specified for M and the behavior deduced for the interconnection of submodules. Mathematical concepts must be chosen so that all properties of interest can be specified and yet the equivalence proofs are still intellectually manageable.

The fixed-point theory of continuous functions has provided a satisfactory framework for specifying and verifying properties of arbitrary interconnections of determinate packet modules. Ellis has demonstrated that acyclic interconnections of determinate and non-determinate packet modules can be studied using the mathematics of relations, but has only been able to give an operational semantics for cyclic interconnections of non-determinate modules. Equivalence proofs based on this operational semantics tend to be rather long and tedious. Many packet systems of practical interest, such as data flow processors and packet memory systems, do contain non-determinate modules for efficient resource sharing. To specify and verify these packet systems, we must develop a more general and more elegant theory for non-determinate packet systems. Preliminary investigations by Brock and Ackerman [Bro78-2] have indicated that a theory of equivalence based on relations between input and output histories does not have the appropriate substitution properties for cyclic interconnections of non-determinate packet modules. They have also suggested a direction for further study. Mathematical frameworks for specifying and verifying non-determinate packet systems bear obvious relationship to semantic theories for data flow programs. Research in these two areas will be mutually supportive.

Fault Tolerance

A packet system consists of a large number of concurrently operating units. Software implementation of fault tolerance is a formidable programming task which can adversely affect fault coverage. Hardware implementation of fault tolerance is thus preferred. We will continue our investigation in hardware fault tolerance techniques for packet systems, using the fault model we have developed. Classical fault tolerance techniques are developed for synchronous hardware systems. Fault detection and masking are synchronized with data processing activities by timing signals, which are either assumed to be failure-free or generated by a fault tolerant clock. In an asynchronous mode of operation, fault tolerance mechanisms must also deal explicitly with synchronization failures in hardware modules. An extremely fruitful area of research is studying techniques for converting well known fault tolerance methods for use in packet systems. We must also study fault tolerance techniques for packet systems incorporating arbitration, the primary means for hardware resource allocation.

We have also identified two special classes of packet systems for fault tolerance considerations: routing networks and a pool of identical processing units. These appear to be commonly occurring subsystems in packet communication computer architectures and hence their practical fault tolerance implementation merits special attention. These two classes of packet systems provide a testing ground for incorporating fault-masking and fault-detection techniques into practical fault tolerant subsystems.

Performance Analysis of Packet Systems

The performance of a particular data flow machine on a given program will be limited either by the throughput capacity of the routing networks and processing units, or by the data dependencies of the program. Thus the performance analysis of a program for execution on a data flow machine has two parts: analysis of data dependencies in the program to determine the constraints imposed on computation rate by program structure; and evaluation of whether the desired computation rate is supported by the routing networks and processing units. Our analysis of the FFT for the Form I machine illustrates this procedure. The first part, analysis of limits on computation rate from data dependencies, has been done using timed Petri nets. This technique must be extended to permit methodical analysis of programs using data structure operations supported by our Structure Processor. For the second part, the primary open problem is analysis of throughput and transit time for various routing network structures.

Work on routing networks has focused around several proposed structures. One possible design for a routing network has alternating stages of switches and arbiters joined by FIFO buffers. We have found that with the proper selection of buffer size this design can be used to construct a large network which has a high average performance for random input. Another design which has a similar structure but uses a more complicated switching unit has also been studied. In this design, the route a particular packet takes is a function of the overall network traffic pattern. This design is interesting since it requires only O(N log N) modules to construct a N input network, and our work indicates that such a network will have a high average performance independent of its size. The goal of our proposed work in this area is the development of valid models for a number of network designs, including the two above. These models should allow us to further refine the designs to improve their efficiency. We plan to check the accuracy of our models by comparing their predictions to results obtained by simulating the actual networks.

4.3 Semantics

We propose to continue our work toward finding the best linguistic constructs for representing data flow computations and for specifying hardware systems employing packet communication. The focus of this work will be on deciding how support for streams and nondeterminate computation should be added to our language VAL.

We also hope to contribute to one of the most pressing problems of computer science; finding the right mathematical foundation for the semantics of nondeterminate programs and systems. An elegant semantic characterization of determinate systems as mapping input histories into output histories has been defined by Kahn [Kah74] and Kosinski [Kos75], and is a straightforward application of Scott's work [Sco76]. A theory of similar elegance for non-determinate systems has not been found. However, we have demonstrated that non-determinate systems may not be characterized by a naive extension of Kahn's theory in which systems are represented by mappings from input histories into sets of output histories [Bro78-2]. One approach has been explored by Kosinski [Kos77, Kos79]. Presently, we are defining a theory of non-determinate computation in which systems are characterized by sets of scenarios, pairs of input histories and output histories ordered by a causality relation. Eventually, this theory will be used to define the semantics of non-determinate data flow languages.

4.4 Implementation Schemes

The design of a Form 4 general purpose data flow machine requires the development of implementation schemes that encompass a complete set of programming concepts including data structures, procedures, and streams.

Data Structures

One implementation scheme for data structures in a Form 2 data flow computer has been developed by Ackerman as mentioned earlier. This work represents one point in a design space that is as yet largely unexplored. Arrays and records are represented by binary trees to permit storage in small units of uniform size that may be allocated anywhere in the physical memory. This choice provides good support for efficiently constructing partially modified structures created by compositions of append operations, but has disadvantages in access time and storage requirements. A better scheme may be to store the representation of each array or record in a localized portion of the physical memory so that local references may be used within the representation, thus conserving space. Other aspects of our Structure Processor concepts needing study are: Should the top level memory modules each serve as a cache for a portion of or for the entire contents of the memory system? What mechanism should be used to derive data streams from arrays to support pipelined computation? What are good ways of applying current and forthcoming device technologies to realizing the Structure Processor?

Procedures and Streams

A good implementation for procedures, streams of data [Wen75], and the **forall** construct is needed for a general purpose data flow computer. In this work [Wen79] the feasibility of extending data flow concepts to support these constructs is being studied. Procedures are necessary to support large computations and are the basis for generalizing data flow architecture concepts to apply to computer systems that serve communities of users. The notion of streams extends the data flow semantics to include the class of computations which are history sensitive in the sense that the behavior of a computation is characterizable as a function from sequences of input values to sequences of output values. This form of computation has been conventionally expressed in languages which either have side-effects or require explicit synchronization primitives. One important characteristic of computations expressed with streams is that the inherent concurrency is not lost, yet they are guaranteed determinate if no explicit non-determinate primitives are used. The **forall** constructs are intended for expressing concurrent operations on data structures. Since this form of concurrency is very often found in many numerical applications, they are useful language features.

One form of data flow processor, which will be studied, uses recursive data flow schemas (represented by acyclic directed graphs) as the basic model of computation. This has several advantages over a processor that supports cyclic data flow schemas: the absence of cycles makes the acknowledge signals required in the Form 2 architecture unnecessary, and use of recursive data flow schemas enhances asynchrony of computation. A satisfactory implementation of procedures, stream data and forall-based computation on this form of data flow processor requires solutions to several problems. An appropriate mechanism for procedure activation for use on this form of machine is required. Similarly, an acyclic representation for recursive procedures is needed. Techniques must be developed for allocating and supporting resources for procedure activations. Support for the stream data type requires both the definition of data flow operators for expressing stream computations and the development of efficient mechanisms for performing those computations. Finally, support for nondeterminate computation is required, for example by implementing the nondeterminate merge operation for streams.

Hardware support of these implementation schemes will require extending of two important subsystems of the Form 2 machine. The instruction memory must support a logical address space which is much larger than that of the Form 2 architecture, and the packet memory must support efficient storage of large data structures whose components may be accessed at significantly different rates.

5. References

[Ack77] Ackerman, William B. A Structure Memory for Data Flow Computers. M.I.T., Laboratory for Computer Science, LCS/TR-186. Cambridge, Ma., September 1977.

[Ack79] Ackerman, William B. and Dennis, Jack B. VAL -- A Value - Oriented Algorithmic Language: Preliminary Reference Manual. M.I.T., Laboratory for Computer Science, Computation Structures Group. In preparation.

[Ami77] Amikura, Katsuhiko. A Logic Design for the Cell Block of a Data-Flow Processor. M.I.T., Laboratory for Computer Science, LCS/TM-93. Cambridge, Ma., December 1977.

[Arv79] Arvind; Gostelow, K. P.; and Plouffe, W. An Asynchronous Programming Language and Computing Machine. University of California - Irvine, Department of Information and Computer Science, TR-114a. Irvine, Ca., January 1979.

[Ash77] Ashcroft, E. A. and Wadge, W. W. "Lucid, a Nonprocedural Language with Iteration." Communications of the ACM, Vol. 20 No. 7 (July 1977), 519-526.

[Avi61] Avizienis, A. "Signed-Digit Number Representations for Fast Parallel Arithmetic." IRE Transactions on Electronic Computers, Vol. EC No. 10 (1961), 389-400.

[Avi64] Avizienis, A. "Binary Compatible Signed-Digit Arithmetic." Proceedings of the FJCC, 1964, 663-671.

[Bou78] Boughton, George A. "Routing Networks in Packet Communication Architectures." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

[Bro78-1] Brock, J. Dean. Operational Semantics of a Data Flow Language. M.I.T., Laboratory for Computer Science, LCS/TM-120. Cambridge, Ma., September 1978.

[Bro78-2] Brock, J. Dean. and Ackerman, William B. An Anomaly in the Specifications of Nondeterminate Packet Systems. M.I.T., Laboratory for Computer Science, Computation Structures Group, Note 33-1. Cambridge, Ma., January 1978.

[Dav78] Davis, A. L. "The Architecture of DDMI: A Recursively Structured Data Driven Machine." Proceedings of the Fifth Annual Symposium on Computer Architecture, Computer Architecture News, Vol. 6 No. 7 (April 1978), 210-215.

[Den75-1] Dennis, Jack B. and Misunas, David P. "A Preliminary Architecture for a Basic Data-Flow Processor." The Second Annual Symposium on Computer Architecture: Conference Proceedings, January 1975, 126-132.

[Den75-2] Dennis, Jack B. "Packet Communication Architecture." Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing, August 1975, 224-229.

[Den77-1] Dennis, Jack B.; Misunas, David P.; and Leung, Clement K. C. A Highly Parallel Processor Using a Data Flow Machine Language. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 134. Cambridge, Ma., 1976.

[Den77-2] Dennis, Jack B. and Weng, Kung-Song. "Application of Data Flow Computation to the Weather Problem." Proceedings of the Symposium of High Speed Computer and Algorithm Organization. New York: Institute of Electrical and Electronics Engineers, 1977.

[Ell77] Ellis, David J. Formal Specifications for Packet Communication Systems, M.I.T., Laboratory for Computer Science, LCS/TR-189. Cambridge, Ma., November 1977.

[Fer78] Feridun, Arif M. "Design of an On-Line Byte-Level Pipelined Arithmetic Processor." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1978.

[Gur77] Gurd, J. and Watson, I. "A Multilayered Data Flow Computer Architecture." Proceedings of the 1977 International Conference on Parallel Processing. August 1977, 94.

[Kah74] Kahn, G. "The Semantics of a Simple Language for Parallel Programming." Proceedings of the IFIP Congress 74. 1974.

[Kos75] Kosinski, P. R. "Mathematical Semantics and Data Flow Programming." Conference Record of the Third ACM Symposium on Principles of Programming Languages, January 1976, 95-103.

[Kos77] Kosinski, Paul R. "A Straightforward Denotational Semantics for Non-Determinate Data Flow Programs." Proceedings of the 5th Annual Symposium on Principles of Programming Languages. 1977.

[Kos79] Kosinski, Paul R. "Data Flow Programs and Implementations: A Mathematical Semantics." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1979.

[Leu75] Leung, Clement.; Misunas, David P.; Neczwid, A.; and Dennis, Jack B. "A Computer Simulation Facility for Packet Communication Architecture." Third Annual Symposium on Computer Architecture: Conference Proceedings. January 1976, 58-63.

[Leu79-1] Leung, Clement. ADL: An Architecture Description Language for Packet Communication Systems. M.I.T., Laboratory for Computer Science, Computation Structures Group. In preparation.

[Leu79-2] Leung, Clement. "Fault Tolerance in Packet Communication Computer Architecture." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1979.

[McC60] McCarthy, J. "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1." Communications of the ACM, Vol. 3 (April 1960), 184-195.

[McN78] McNally, Mary E. "The Design of an Arbitration Network for a Data-Flow Processor." M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 164. Cambridge, Ma., July 1978.

[Mir77] Miranker, Glen S. "Implementation of Procedures on a Class of Data Flow Processors." Proceedings of the 1977 International Conference on Parallel Processing. Institute of Electrical and Electronics Engineers, August 1977, 77-86.

[Mis78] Misunas, David P. A Computer Architecture for Data-Flow Computation. M.I.T., Laboratory for Computer Science, LCS/TM-100. Cambridge, Ma., March 1978.

[Mon79] Montz, Lynn B. "Safety and Optimization Transformations for Data Flow Programs." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1979.

[Pat78] Patil, Suhas S.; Keller, R. M.; and Lindstrom, G. An Architecture for a Loosely-Coupled Parallel Processor (Draft). University of Utah, Department of Computer Science, UUCS-78-105. Salt Lake City, Utah, July 1978.

[Ram73] Ramchandani, Chander. "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, September 1973.

[Sco76] Scott, Dana S. "Data Types as Lattices." SIAM Journal of Computing, Vol. 5 No. 5 (September 1976), 522-587.

[Syr77] Syre, J. C.; Comte, D.; and Hifdi, N. "Pipelining, Parallelism and Asynchronism in the LAU System." Proceedings of the 1977 International Conference on Parallel Processing, August 1977, 87-92.

[TI79] Texas Instruments. Private communication. January 1979.

[Tre77] Treleaven, P. C. Principal Components of Data Flow Computers. University of Newcastle upon Tyne, Computing Laboratory, TR-108. Newcastle upon Tyne, England, July 1977.

[Tri77] Trivedi, K. and Ercegovac, M. "On-Line Algorithms for Division and Multiplication." IEEE Transactions on Computers, (July 1977), 681-687.

[Wen75] Weng, Kung-Song. Stream-Oriented Computation in Recursive Data Flow Schemas. M.I.T., Laboratory for Computer Science, LCS/TM-68. Cambridge, Ma., 1975.

[Wen79] Weng, Kung-Song. "An Abstract Implementation of a Generalized Data Flow Processor." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1979.

6. Resume of Principal Investigator

Jack B. Dennis

Personal

Born October 13, 1931, in Elizabeth, New Jersey

Graduate (1949) of Darien High School, Darien, Connecticut

Graduate of Massachusetts Institute of Technology,
Department of Electrical Engineering

Bachelor and Master of Science	June 1954
Doctor of Science	September 1958
Dissertation: "Mathematical Programming and Electrical Networks"	

Employment

Air Force Cambridge Research Laboratories — 1951-1954
Design and testing of electronic systems (MIT Co-op program)

Bell Telephone Laboratories — Summer 1954
Development of electronic switching equipment

M.I.T. Department of Electrical Engineering — 1954-1958
Graduate Assistant
Taught most classical Electrical Engineering subjects -- circuits, electronics, field theory
Research in mathematical programming including preparation of computer programs for gradient optimization methods and for the transportation problem

Member of M.I.T Faculty, Department of Electrical Engineering — 1958 - Present

Appointed full professor beginning July 1969

Professional Experience

As a faculty member, Dr. Dennis developed four M.I.T. subjects in the areas of computer theory and computer systems:

1. Theoretical Models for Computation.
[Finite state machines, pushdown automata, Turing machines, and the corresponding classes of abstract language.]

2. Computation Structures (basic undergraduate subject in Computer Science)
[The relation of hardware and software technology to the implementation of programming languages.]

3. Structure of Computer Systems
[Study of advanced concepts in computer architecture and operating systems.]

4. Semantic Foundations for Computer Systems
[Formal semantics and its application to design of computer hardware/software systems.]

Professor Dennis led a group that designed and implemented one of the earliest time-shared computer installations. He joined M.I.T.'s Project MAC at its inception in 1963 and participated in specifying advanced hardware embodying the concept of segmentation for Multics. He now leads the Computation Structures Group of the M.I.T. Laboratory for Computer Science.

Professor Dennis directs graduate research in "computation structures" with the goal of evolving useful theoretical approaches to problems derived from experience with practical computer systems. Twelve doctoral theses have been written under his supervision.

Professor Dennis has served several major computer manufacturers as a consultant on questions of computer system organization.

Professional Activities

Dr. Dennis is a member of the Association of Computing Machinery and its Special Interest Group on Operating Systems, Programming Languages and Computer Architecture. He is a member of the Computer Group of the IEEE and was elected Fellow of the IEEE. He was responsible for organizing the first ACM Symposium on Operating System Principles and workshop conferences on Concurrency in Computation and on Semantic Foundations for Structured Programming.

Dr. Dennis was elected to membership in Eta Kappa Nu, Tau Beta Pi and Sigma Xi.

Patents

No. 3,962,706. Dennis, J. B., and Misunas, D. P. Data Processing Apparatus for Highly Parallel Execution of Stored Programs.

Publications

Dennis, J. B. "A High-Speed Computer Technique for the Transportation Problem." J. of the ACM. April 1958.

Dennis, J. B. "System Synthesis with the Aid of Computers." Communications and Electronics. November 1959.

Dennis, J. B. Mathematical Programming and Electrical Networks. John Wiley and Sons, Inc., N. Y., 1959.

Dennis, J. B. "Computer Control of an Analog Vocal Tract." Speech Communication Seminar. Stockholm, August 1962.

Dennis, J. B. "The Use of Computers in Speech Research." Annals of the New York Academy of Sciences. Computers in Medicine and Biology. July 1964.

Dennis, J. B. "A Multi-User Computer Facility for Education and Research." Comm. of the ACM, Vol. 7 No. 9 (September 1964).

Dennis, J. B. Program Structure in a Multi-Access Computer. M.I.T., Laboratory for Computer Science, LCS/TR-11. Cambridge, Ma., December 1964.

Dennis, J. B. and Glaser, E. L. "Structure of On-Line Information Processing Systems." Proceedings of the Second Congress on Information System Sciences. Spartan Books, January 1965.

Dennis, J. B. "Distributed Solution of Network Programming Problems." IEEE Trans. on Communications Systems, Vol. CS-12 No. 2 (June 1964).

Dennis, J. B. "Segmentation and the Design of Multi-Programmed Computer Systems." J. of the ACM, Vol. 12 No. 4 (October 1965).

Dennis, J. B. and Van Horn, E. C. "Programming Semantics for Multi-Programmed Computations." Comm. of the ACM, Vol. 9 No. 2 (February 1966).

Dennis, J. B. "A Position Paper on Computing and Communications." Comm. of the ACM, Vol. 11 No. 5 (May 1968).

Dennis, J. B. and Daley, R. C. "Virtual Memory, Processes, and Sharing in Multics." Comm. of the ACM, Vol. 11 No. 5 (May 1968).

Dennis, J. B. "Future Trends in Time-Sharing Systems." Time-Sharing Innovation for Operations Research and Decision-Making. Washington Operations Research Council, 1969.

Dennis, J. B. The Structure of the Computer Utility. Presented at the Joint IBM/University of Newcastle Upon Tyne Seminar held at the Computing Laboratory, University of Newcastle Upon Tyne, September 1969.

Dennis, J. B. "Programming Generality, Parallelism and Computer Architecture." Information Processing 68. North-Holland Publishing Co., Amsterdam, 1969.

Dennis, J. B. "Modular, Asynchronous Control Structures for a High Performance Processor." Record of the Project MAC Conference on Concurrent Systems and Parallel Computation. ACM, New York 1970.

Dennis, J. B. and Patil, S. S. "Speed Independent Asynchronous Circuits." Proceedings of the Fourth International Conference on System Sciences. Western Periodicals Co., January 1971.

Dennis, J. B. "Coroutines and Parallel Computation." Proceedings of the Fifth Annual Princeton Conference on Information Sciences and Systems. March 1971.

Dennis, J. B. "On the Design and Specification of a Common Base Language." Proceedings of the Symposium on Computers and Automata. Polytechnic Press of the Polytechnic Institute of Brooklyn, New York, April 1971.

Dennis, J. B. and Patil, S. S. "The Description and Realization of Digital Systems." Sixth Annual IEEE Computer Society International Conference, Digest of Papers. IEEE, New York 1972.

Dennis, J. B. "The Design and Construction of Software Systems." Advanced Course on Software Engineering, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1973, 12-28.

Dennis, J. B. "Concurrency in Software Systems." Advanced Course on Software Engineering, Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 1973, 111-127.

Dennis, J. B. "Modularity." Advanced Course on Software Engineering, Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 1973, 128-182.

Dennis, J. B. and Patil, S. S. "The Description and Realization of Digital Systems." Revue Francaise d'Automatique, Informatique et de Recherche Operationnelle, February 1973.

Dennis, J. B. "Semantics of Languages and Systems." Report of Workshop 2, Proceedings of a Symposium on the High Cost of Software. Stanford Research Institute, Menlo Park, Ca., September 1973.

Dennis, J. B. Report on ACM-SIGPLAN-SIGOPS interface meeting, semantics session. SIGPLAN Notices, Vol. 8 No. 9 (September 1973).

Dennis, J. B. "First Version of a Data Flow Procedure Language." Programming Symposium: Proceedings, Colloque sur la Programmation. Lecture Notes in Computer Science, Vol. 19. Springer-Verlag, 1974.

Dennis, J. B. and Misunas, D. "A Computer Architecture for Highly Parallel Signal Processing." Proceedings of the ACM Annual Conference. 1974, 402-409.

Dennis, J. B.; Fosseen, J. B.; and Linderman, J. P. "Data Flow Schemas." International Symposium on Theoretical Programming, Lecture Notes in Computer Science, Vol. 5. Springer-Verlag, 1974, 187-216.

Dennis, J. B. and Misunas, D. "A Preliminary Architecture for a Basic Data-Flow Processor." Proceedings of the 2nd Annual Symposium on Computer Architecture, IEEE, New York, 1975, 126-132.

Dennis, J. B. "Packet Communication Architecture." Proceedings of the 1975 Sagamore Computer Conference on Parallel Processing. IEEE, New York, August 1975.

Dennis, J. B. "An Example of Programming with Abstract Data Types." SIGPLAN Notices, Special Issue on Programming Language Design. July 1975.

Leung, C. K. C.; Misunas, D. P.; Neczwid, A.; and Dennis, J. B. "A Computer Simulation Facility for Packet Communication Architecture." Proceedings of the 3rd Annual Symposium on Computer Architecture. IEEE, New York 1976, 58-63.

Dennis, J. B. "Computer Architecture and the Cost of Software." Presented at the Third Annual Computer Architecture Symposium. It was published in SIGARCH News, April 1976.

Dennis, J. B. and Weng, K.-S. "Application of Data Flow Computation to the Weather Problem." Proceedings of the Symposium on High Speed Computer and Algorithm Organization. IEEE, 1977.

Dennis, J. B. "A Language Design for Structured Concurrency." Workshop on Design and Implementation of Programming Languages, Lecture Notes in Computer Science, Vol. 54. Springer-Verlag, 1977, 231-242.

Dennis, J. B.; Fuller, S. H.; Ackerman, W. B.; Swan, R. J.; and Weng, K.-S. "Research Directions in Computer Architecture." The Impact of Research on Software Technology (P. Wegner, Ed.). M.I.T. Press, forthcoming.

Bryant, R. E. and Dennis, J. B. "Concurrent Programming." The Impact of Research on Software Technology (P. Wegner, Ed.). M.I.T. Press, forthcoming.