Final Report

Data Flow Computer Architecture

Jack B. Dennis

Professor of Computer Science and Engineering

October 1987

Advanced Research Projects Agency

Full research support	September 1963 to March 1975
Partial support	April 1975 to December 1985

National Science Foundation

Research Grant DCR74-21892	April 1975 to September 1977
Research Grant DCR75-04060	April 1975 to December 1979
Research Grant MCS-7915255	March 1980 to November 1985

University of California
Lawrence Livermore National Laboratory

Subcontract 8545403	January 1978 to September 1979

Department of Energy

Contract DE-AC02-79ER10473	July 1979 to December 1985

National Aeronautics and Space Administration
Ames Research Center

University Interchange NCA2-OR425-101	February 1981 to April 1982
Research Grant NAG 2-247	July 1983 to October 1985

Masssachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, MA 02139

Table of Contents

1 Introduction

This report covers the work done by the Computation Structures Group of the MIT Laboratory for Computer Science on developing models, languages, and architectures for data flow computation from 1966 to the end of 1985. The work was supported by research grants and contracts from NSF, the University of California, DOE, NASA, and DARPA having periods of support as follows:

Advanced Research Projects Agency (DARPA)

Full research support
1 September 1963 to 31 March 1975

Principal results:

- Contributions to conceptual design of multiple processor and interactive computer systems.
- Addressing schemes for time-shared, multiprocess computer systems; the concept of *capability-based architecture.*
- The *working set* model for storage management in multiprocess computer systems.
- Extension of the *banker's algorithm* for deadlock-free sharing of resources.
- Conception of *data flow program graphs* and their evolution into a complete program model.
- First *proof of determinacy* for a system of cooperating computational processes; its extension to data flow schemas.
- Early proposals for *data flow computers*: the static data flow architecture and a multiprocessor based on allocation of processors to procedure activations.
- The *base language* concept and the use of operational models to establish computer system correctness; correctness proofs for nondeterministic operational models.
- Early investigation of the use of *streams* as a means for functional expression of input/output processes.

Occasional partial support
1 April 1975 to 31 December 1985

National Science Foundation (NSF)

Research Grant DCR74-21892
Semantic Foundations for Structured Programming
1 April 1975 to 30 September 1977

Principal results:

- Operational semantic techniques for system analysis and design.

Research Grant DCR75-04060 (MCS-7504060)
Data Flow Computer Architecture

1 April 1975 to 31 December 1979

Principal results:

- Contributions to Petri net theory and applications.
- Design methodologies for self-timed systems.

Research Grant MCS-7915255 (DCR-7915255)
Data Flow Computer Architecture

1 March 1980 to 30 November 1985

Principal results:

- Contributions to the theory and practice of packet_routing networks for processor interconnection.
- Elucidation of the principles of *packet communication architecture*; development of the *PADL* architecture description language.
- *Fault tolerance* techniques for data flow computers.
- Methodology for proving compositions of modules specified as transformers of streams.
- Study of *functional data structures*; semantic modeling and the *packet memory* concept for data structures.
- The observation that functional languages can be implemented without use of *cyclic data structures*, and therefore may employ *reference count* storage management and be more compatible with concurrent execution.
- The *scenario theory* for the semantics of compositions of nondeterminate program modules.

- Extensions of *type-inference* methods and *value-binding* mechanisms for *functional programming environments.*

University of California
Lawrence Livermore National Laboratory

Subcontract 8545403
Development and Application of Data Flow Computers
17 January 1978 to 30 September 1979

Principal results:

- Design of the programming language Val.
- Translator/interpreter implementation of Val.
- Analysis of a simplified hydrodynamics code—Simple.

Department of Energy (DOE)

Contract DE-AC02-79ER10473
Data Flow Computer Architecture

1 July 1979 to 30 June 1982
1 September 1982 to 31 March 1984
1 June 1984 to 31 December 1985

Principal results:

- Translation of the Simple code into Val; analysis of Simple.
- Construction of a *data flow engineering model.*
- Experimental back end for the Val compiler.
- Structural analysis and simulation of packet-routing networks.
- Trial design and fabrication of key VLSI components for static data flow computers.
- Development of program structuring principles for partial differential equation codes.
- Development of diagnostic techniques for packet-routing networks.
- Study of the particle-in-cell method for plasma simulation.

National Aeronautics and Space Agency (NASA)

Ames Research Center, Moffett Field CA
University Consortium Interchange NCA2-OR425-101

High Speed Data Flow Computer Architecture
for the Solution of the Navier-Stokes Equations
1 February 1981 to 30 April 1982

Research Grant NAG 2-247
Specification of a Data Flow Computer for Aerodynamic Simulation
1 July 1983 to 31 October 1985

Principal results:

- Translation of ARC3D, a three-dimension aerodynamic simulation code, into Val.
- Mapping of aerodynamic simulation onto a data flow computer with 256 processing elements.
- Refinement of the static architecture to meet requirements of benchmark codes.
- Comparative evaluation of standard and cyclic reduction methods for solution of block tridiagonal equation systems in the context of data flow computation.

Certain of the written proposals for these grants and contracts have been published [99, 104, 105, 108].

The entire research program on data flow models, languages, and architecture has been carried out under the direction of Professor Jack B. Dennis. The program contributed papers to refereed journals, several book chapters, and many conference papers and reports. The project supported many students who earned 28 doctorates, 42 master's degrees and 38 bachelor's degrees.

The research led to major projects at Manchester University, the University of Utah [197], Texas Instruments [63], Hughes Aircraft, and Loral Instrumentation. There has been much interest in Japan, including government-sponsored projects and commercial products. Many other workers have studied a variety of specialized aspects of data flow computation, in particular at Iowa State University [256] and at the University of Southwestern Louisianna.

Research on data flow concepts at MIT was broadened and strengthened in 1979 by the appointment of Professor Arvind. His work, begun at the University of California at Irvine, added a new dimension to the field in the *unravelling interpreter* and the *tagged-token* data flow architecture.

The application of data flow principles to large-scale scientific computation is now being developed for commercial use by Dataflow Technology Corporation, a company founded by Professor Dennis.

2 Background

When Jack B. Dennis joined the MIT faculty in Electrical Engineering in 1958, he had already accumulated four years of experience working with computer systems. As a student in the electrical engineering VI-A co-op program, he improved the design of encoding circuits that prepared radar data for use by an early aircraft tracking computer at the Air Force Cambridge Research Center. In his MIT graduate work in operations research, he developed Whirlwind programs for several problems, including an unusually efficient code for the transportation problem, a specialized linear programming application, that was extensively used by a sponsoring company.

As a young faculty member, Professor Dennis was responsible for the operation and improvement of the TX-0 computer, an early high-speed transistorized machine built by the MIT Lincoln Laoratory and loaned to the Electrical Engineering Department. He designed and directed installation of extensions to the machine's instruction set and additions to its collection of peripheral equipment. He also developed software for the TX-0, including a macro assembler which evolved into a commercial assembly language, and an interactive debugging program that led to later interactive debugging tools.

In 1960 the efforts of John McCarthy, J.C.R. Licklider, Herbert Teager, and others had generated great interest in the concept of *time-shared* operation of a computer system to support interactive use by programmers and users [144]. The gift in 1961 of a PDP-1 computer by the Digital Equipment Company to the MIT Electrical Engineering Department offered the opportunity to apply the time-sharing concept on a small scale for a unique community of users desiring hands-on use of a computer: electrical engineering students and the staff of the Research Laboratory of Electronics. With the help of students and a small staff, Professor Dennis designed and implemented hardware changes and software that turned the PDP-1 into a unique facility especially suited to work where a tight coupling of the computer to an experimental set-up was required [92]. It was one of the first three interactive, time-shared computers to become operational—all in Cambridge, Massachusetts and all in response to McCarthy's advocacy.

The demonstrated effectiveness of interactive computing led to the formation at MIT of the Long Range Computer Study Group to recommend how these ideas could be implemented on a large scale. The study, in which Professor Dennis participated, considered the concepts and technology needed to create efficient computer structures and operating systems that would support productive, interactive, time-shared computing for a large community of users such as an entire university. The committee recommended multiprocessor architecture for such an ambitious system, noting the requirements for expandability, support for concurrency, reliability,

and efficiency of resource allocation [295]. The work of the committee led directly to MIT's Project MAC [143] and development of the Multics time-sharing system [64, 63].

3 Computer System Concepts

In the fall of 1963 Professor Dennis joined Project MAC to participate in formulating plans for the Multics system, and to lead a new research group in study of the long-range problems of computer system structure posed by the new style of computer use. One area of intense investigation concerned how information—units of data and program—should be organized, accessed, and updated in the context of the envisioned "computer utility". The initial work of the group, done in collaboration with the Multics implementation team, involved unifying the notions of virtual memory, paging, and descriptor-based addressing, and finding an appropriate concrete realization of these concepts for use in Multics. The first results of these studies were published in a widely-read report [80, 81], and their application in Multics is discussed in [67, 68]. This work led to Dennis' election as Fellow of the Institute of Electrical and Electronic Engineers. A paper projecting the likely form of the future computing utility was also published by Dennis [82, 83]. Although the ideas presented in this paper have taken substance within the framework of "distributed computing systems," the concept of system *operator* has yet to be established with the strength envisioned.

Over the next few years the Dennis group made many contributions to the conceptual foundations for the design of computer systems. The viewpoint espoused in this effort was that computer hardware and an operating system jointly define the interface upon which useful application software is built. Hence, the design of the hardware and operating system must be coordinated so that the most desirable interface is obtained, consistent with practical constraints.

Two major contributions were the concepts of "capability" and "sphere of protection" introduced in a well-known paper by Dennis and van Horn [136, 135]. This work led directly to work on computers using "capability-based addressing", specifically the Chicago Magic Number Computer and the Plessey 250. An excellent review of these projects and the principles of capability architecture is given by Fabry [142]. Many of the ideas tested in these projects were subsequently incorporated in the IBM System 38 architecture [185].

Other subjects explored include issues of input/output control [291, 290], subsystem sharing [278], the management of names for information [88], and the control of information-sharing [304] in multiprocessor computer systems employing segmented addressing. Hebalkar's doctoral thesis [176, 177, 175] extended the scope of known methods for the coordinated sharing of resources with a guarantee of deadlock-free operation.

The group's work on the conceptual basis of time-shared computer systems led to study of related issues of performance modeling and analysis, primarily by Peter Denning [70, 71, 72]. In

his doctoral research, Denning developed the *working set* principle into a practical model for storage management in multi-process computer systems [73, 74, 75]. This work became widely known and led to an important joint paper with Aho and Ullman [303].

Professor Dennis was invited to give lectures on these subjects at an "Advanced Course in Software Engineering" held in Munich [96, 89, 90, 91], and also participated in a workshop on "Computer Architecture and the Cost of Software" [100, 101].

4 The Origins of Data Flow Concepts

The work on computer systems issues led us to recognize that the interface defined by the hardware and operating software of a computer system is a computer language—a language made up of all the facilities provided by the system for users to implement application codes. We call this set of facilities the *base language* of the computer system. The base language is far more extensive than the usual notion of a programming language. It includes all facilities for addressing, for scheduling, for accessing and manipulating files, and for controlling concurrency and directing input and output activities. An important observation was that the complexities of the base language of a computer system are responsible for a large portion of the difficulties of large software projects.

The idea arose naturally that it would be better to design the base language consciously rather than let the base language arise *ad hoc* from the fortuitous independent decisions of hardware and software designers. The system architecture should be in such harmony with high-level user languages that the relationship between the two is simple, direct, and complete. We set as our goal the formulation of the simplest base language that would serve the needs of computer users, and the development of computer architectures that would efficiently support such a simplified basis for general-purpose computing.

The first steps were evolutionary—a suggestion that a straightforward combination of ideas already explored would suffice to achieve the goal. The concept of capability-based addressing mentioned above was the principal achievement of this first phase.

The next step was revolutionary—the suggestion that fine grain concurrency must be exploited in a computer system design if the requirements of modular programming are to be met efficiently. This thesis was put forth at the 1968 IFIP Congress in Edinburgh [84]. *Data flow program graphs* (see below), in which nodes represent individual operators of a program, turned out to be the right program model for realizing this suggestion.

The concurrency of activities plays an important role in real computing problems. Input/output transactions inherently involve the notion of synchronized, independent activities. Several users communicating with a time-shared computer run processes that interact with each other and make concurrent demands for computing resources. Yet, computer architects and designers of programming languages had not responded to the need for simple and elegant means of expressing concurrent computations and for efficient means for supporting their execution.

The problem was that the only avenues of development open without giving up strongly ingrained traditions of computer architecture were two: independent computational processes sharing an address space and communicating by means of "synchronizing primitives"; and independent computing elements with no shared memory communicating by "message-passing".

In 1966 Jorge Rodriguez, a graduate student working with Mr. Douglas Ross in the MIT automated machine tool project, proposed a novel idea to Professor Dennis: Why not use a directed graph model for programs in which the arcs denote data value production/consumption relationships between operator nodes? Rodriguez thought that such a graph model would be useful in optimizing compilers, an idea that is just now entering compiler practice [311]. Dennis found the idea to be just the concept needed to fulfill the goals of a base language.

5 Computational Models

A cornerstone of the research program of the Computation Structures Group has been the basing of computer system architecture on sound models of computation. The first such model was presented in the Dennis and van Horn paper on semantics [136]. Although informal, this paper sketched what would later be known as an *operational semantics* for an abstract computer system.

Rodriguez' work [280, 281] was the first formal parallel computation model that incorporated data flow and sequencing control in one and the same graph. A similar model emphasizing the semantics of function call and return was formulated in Suhas Patil's master's thesis [260].

It was satisfying to note certain similarities in other contemporary work: the work on parallel program schemas by Karp and Miller [195, 196] at the IBM Research Center; the report by Presberg, Saint and Shapiro [270] at Massachusetts Computer Associates; the thesis of Burt Sutherland [294] on a graphical editing tool for programs; and the Stanford University thesis of Duane Adams [11].

Several years of exploration led to the formal exposition of the data flow program graph model as it is generally known. The first definitive description of the model was in the master's thesis of John Fosseen [147]. The MIT work inspired a similar formulation by Paul Kosinsky, then at the IBM Research Center [198].

The basic data flow model, extended to include functional data structures, was described by Dennis in 1974 [93]. This paper also contrasted alternative ways of dealing with recursive function invocation and introduced the concept of an unchanging heap for holding data structures. We noted [94] that storage management for the underlying program execution model for data flow program graphs could be accomplished using the reference count scheme, even when functions are passed as arguments or results of function activations. This fact, even though it appeals to us as very promising for efficient parallel implementations of functional languages, seemed little appreciated at the time.

A major insight gained during our research on computational models was the realization that asynchronous parallel computing could be accomplished without risking the hazards of timing inherent in conventional approaches to structuring parallel programs. Although this fact is now widely appreciated, it was a surprising idea in 1966. To the best of our knowledge, Earl van Horn's work [305, 306] provided the first proof that a system of computational processes assembled according to simple rules of construction, is inherently determinate—free of timing hazards. Van Horn's work used the concepts and terminology of conventional multiprocess operating systems.[1] Yet, the ideas transferred naturally to the data flow model. Proofs of determinacy of the data flow model were given by Rodriguez, then more general formulations by Patil [261] and Denning [69] captured the essence of the idea.

The language of data flow graphs was a new kind of parallel program schema, one in which determinacy is guaranteed. Recognizing that data flow schemas are fundamentally different from other schematic models of programs, it was natural to explore their formal classification to rephrase the usual questions about equivalence properties and the like, and to inquire about their decidability. Research on these questions led to a master's thesis [209, 210] and doctoral theses by Luconi [224, 225], Slutz [288, 289], Linderman [222, 221], and Qualitz [271, 272].

The question of deciding equivalence of data flow schemas was addressed by Rodriguez [280] in his work on program graphs, but this early effort was flawed. Later we examined the subject again, narrowed to the question of output equivalence for free data flow program graphs. This problem appears to be decidable [119].

6 Generalizations

We devoted much effort to generalizing our base language model to be a sound and complete semantic model for as wide a class of computing tasks as possible. One important topic was the handling of procedure-valued variables that might occur as inputs to program modules or returned as one result of a module. We analyzed these issues within the framework of the base language ideas of [86] and reported the results in [12, 13].

An area that required much thought to arrive at a satisfactory resolution is the expression of input/output actions in a manner compatible with data flow ideas and the functional programming style. We were aware that coroutines [61, 62, 233] in certain ways express the "semantics" we wished to encompass with an extended data flow formulation, and we had observed a correspondence between data flow graphs and cooperating groups of coroutines [87].

Very early in our study of the data flow model, we had observed the naturalness with which signal processing operations, such as filtering, convolution, etc., fit the model. Since an electrical

[1] The work of Karp and Miller on determinacy [195] was contemporary with and independent of the work of van Horn and Rodriguez.

signal, quantized in time, is essentially an unending sequence of values, we felt that the concept of a *stream* of values was very important in data flow computation. It seemed that certain problems, such as the comparison of "fringes" [179], had elegant formulations only in terms of streams. Kung-Song Weng, in his master's thesis [312, 137], showed how the stream concept should be incorporated into the data flow model to achieve the most desirable semantic model and source language functionality. The concept of using infinite sequences in a model for parallel computation was given early treatment by Gilles Kahn [194].

To complete our base language proposal, the problem of expressing nondeterminate computations required resolution. We had long been aware that certain computer applications, for example a simple airline reservation system [103], were not expressible in a strictly functional programming language. We were also aware of the monitor construct that evolved from the thinking of Dijkstra, Wirth, and Brinch-Hansen [180], an important contribution to dealing with nondeterminacy in the context of imperative-style programming languages. We found that proposals for introducing a closely analogous construct into data flow models not entirely satisfying [15]. Our most advanced proposal is for a data flow *guardian* construct [110] that draws upon ideas of Hewitt [179] and others.

A survey of data flow models was presented in six lectures to the NATO Summer School at Marktoberdorf, West Germany, in 1984 [115].

7 Data Flow Architecture

Our efforts to develop architectural proposals for data flow computers began early in 1972, inspired by the problem of achieving real-time computer synthesis of music. This challenge was posed by Professor Barry Vercoe, who had just arrived at MIT with the goal of establishing a preeminent experimental music studio. Thus, our earliest proposal was for a machine that could execute the restricted sorts of data flow graphs arising in signal-processing applications [128].

At first we thought that this machine concept, in which all actors would fire an equal number of times, would be limited in its range of application. Time has proved otherwise. We soon discovered how mechanisms for handling conditional graphs and iterations could be incorporated into the architecture [129, 130].

These first proposals for data flow machines used many *instruction cells* to hold instructions representing the individual actors of a data flow program graph. The thought was that each cell might be a separate silicon chip. The cells would be served by a small number of specialized, high-performance functional units. Communication between the cells and the functional units would be provided by an *arbitration network* to funnel active operations to the required functional units, and a *distribution network* to pass result packets to their respective target instruction cells. Since a serial representation of information was desirable for the cells but a parallel form was presumed to be needed at the functional units, serial-to-parallel and parallel-

to-serial conversion was called for as packets passed through the two networks. The properties of such networks were studied by Boughton [29], and designs for self-timed implementations of the necessary components were developed [242].

By 1975 we realized that the idea of providing an individual instruction cell for each data flow actor was too extreme. It would be more efficient for many instructions to share the functional logic for recording operand arrivals and recognizing enable conditions. Hence, the notion of a hardware *cell block* was proposed. Since then, the number of instructions to be handled by one processing element of the machine has risen from an initial guess of 16 to the current guess of 4096. At one point, byte-serial transmission of data between the major sections of the machine was envisioned, and a study of byte-serial floating point arithmetic units was done [145, 146].

Later, the proposed machine configuration was expanded to include *array memory* for holding array values in sequences of memory locations in support of large-scale scientific computation. This design has proved to be well matched to important benchmark problems, and a tutorial account of it has been published [106, 107, 109]. Further and more refined descriptions of the architecture of data flow machines for scientific computation have been presented together with our analyses of specific applications. The paper [78] presents a commentary on the attractive features of data flow architecture for advanced supercomputers.

For his doctoral research, James Rumbaugh developed a contrasting kind of data flow computer based on associating processors with activations of procedures [284, 285]. In this machine, an auxiliary memory is arranged to hold function activations that are dormant while waiting for subsidiary activations to terminate, or are active but awaiting their turn for processing. This is one study of a highly parallel architecture that includes a hierarchical memory structure, albeit a "coarse grain" scheme. Rumbaugh's design anticipated important elements that were to appear later in experimental machines. Specifically, the instruction execution system of Rumbaugh's activation processor is an antecedent of the processing elements of the Texas Instruments data flow testbed [65, 193]. Later, related ideas appeared in commercial data flow computers offered by NEC [205], and in a paper design developed by the Hughes Aircraft Company [307, 308].

The problem of function call and termination in a data flow computer was analyzed by Glenn Miranker [237, 240, 241], who proposed a novel implementation scheme for use within the framework of the Dennis-Misunas proposals [129, 130].

Later advances in the static dataflow architecture went hand in hand with our analysis of potential applications for data flow computers. The focus was on applications to large-scale scientific computing. As a result, two major developments took place: an *array memory* capability was added to hold the multi-million-word data bases required by very large numerical problems; and we discovered that pipelined operation of data flow programs offered a means of obtaining very efficient operation of a data flow computer.

The array memory was first mentioned in [113], and later in [123]. We now envision associating one module of array memory with each processing element [79], so non-local array references are implemented using transmission through the routing network.

The merit of pipelined operation of data flow programs became evident as our research turned to the study of practical scientific applications. The form of pipelining considered is one in which the actors or instructions of a data flow program are grouped into stages and sets of data values move through the stages in succession, thereby keeping all stages continuously busy. The first exposition of this idea in application to a scientific application was the study by Dennis and Weng of a global weather model [138]. The idea, which is a stored program variation on "systolic computation" [204], has been used subsequently in all our studies of scientific benchmark codes.

A crucial element of a data flow computer is the mechanism that identifies the instructions available for execution and chooses their order of execution. A comparison of two schemes for implementing this function was made by Marcovitz [227, 228].

The work on data flow under Professor Dennis [128, 129] became known to Professor Arvind at the University of California at Irvine. With students and faculty colleagues, Arvind formulated an alternative model of data flow computation, the *unraveling interpreter*, and a corresponding architecture, the *tagged-token* architecture [16]. On the basis of the originality and soundness of his work, MIT invited Arvind to join its faculty in Cambridge. Since 1979 Arvind has given strength to data flow research at MIT, and his projects have had their own independent evolution [17, 18, 19]. Toshio Shimada, a visitor to the Laboratory for Computer Science, returned to Japan and formed a group at the Electrotechnical Laboratory that has built a practical tagged-token machine, the Sigma 1 [316].

During the 1970s and early 1980s, annual workshops were held for one week each summer at the MIT Endicott estate. These gatherings served to bring together workers from around the world who were interested in discussing and contributing to the technology of data flow computation. Conference notes from some of these meetings have been published [243, 247, 248, 46].

8 Packet Communication Architecture

The various architectural proposals we had devised for data flow computers had a common characteristic: they consisted of hardware units interconnected by links that conveyed information packets from one unit to another. Therefore, we abstracted from these systems to identify useful common properties [95, 97]. For example, that determinate behavior of interconnections of modules is preserved by the communication protocols used in such systems was shown by Patil [261]. Also, means of simulating such systems were developed and the problems of performing these simulations using distributed processing were solved [39, 40].

Our research suggested that a descriptive methodology particularly suited to such systems would have advantages in clarity and simplicity. This led to the design of PADL (for *Packet Architecture Description Language*) [211, 216, 181, 182], which was used to specify several hardware elements of our proposed machine designs.

9 Structure Memory

An important issue in any computational model is the treatment of data structures. To be consistent with the desired side-effect-free character of functional languages, the user should regard data structures as being created but never modified during the progress of computation. Of course, what actually happens in the supporting machine does not matter so long as the effect seen by the user is the effect intended. A conflict needing resolution is that the desired user semantics appears to require much copying of data structures, whereas the desired computation may often be best accomplished by in-place modification of data structures in the machine.

Our first efforts were semantic modeling experiments [84, 50, 148, 139, 186, 187] that convinced us that we should present the functional view of data structures to users. This view is reflected in the extensions to data flow program graphs eventually adopted for expressing the creation and use of data structures [93].

Once the semantic model for data structures had been determined, the search for efficient implementation techniques began. Our efforts concentrated on approaches suitable for use in highly concurrent realizations of data flow computers. The concept of a hierarchical associative memory, introduced in [84], was pursued by Gertz [157, 158].

Later, schemes for implementing the functional data structure heap, as described in [93], were devised in the form of packet communication systems [95, 97]. Using our observation that data flow program graphs can be implemented with a cycle-free heap [94], these schemes used the reference count technique for determining when storage occupied by items may be recovered. They were subsequently elaborated and refined in the master's thesis of Ackerman [2, 3, 4], and in subsequent work by Weng [313, 314].

10 The Engineering Model

About 1977 we began construction of a simulation means for evaluating dataflow architectures. Since we could not foresee obtaining enough simulation time on available laboratory computers, we proposed to build a multiprocessor system specialized for the purpose of simulating packet comunication architectures. A summary paper on this proposal is [217]. Our work on this topic led to significant original work on the problem of distributed simulation by Randal Bryant [39, 40].

We soon found that it would be just as easy to build a multiprocessor system that would

more directly model the behavior of proposed data flow machines. This concept was pursued, leading to construction of the MIT data flow engineering model, as described in [125, 127] which report completion of the project. The experimental system consists of eight processing elements connected together by a packet-routing network. Its design is discussed in [116, 117]. Studies by students contributed to the design of the engineering model in several areas, including structures for packet communication [188, 53] and implementation of floating point arithmetic [301, 302]. A trial design of the processing module was done by Vishniac [309, 310], and several simulation efforts were carried out [277, 236]. The design of the packet-routing network (see below) was augmented with diagnostic facilities that provided for pinpointing a faulty router module without having to dismantle the interconnection network [220].

The data flow instruction set chosen for implementation in microcode is described in a manual written by Ackerman [7], and an interpreter for this instruction set was implemented by Todd [298]. This instruction set served as the target language for subsequent compiler experiments.

Several projects were undertaken to develop trial designs for data flow hardware. Early work was done using the instruction cell concept as a design challenge for self-timed logic design methodology [242]. As the ideas moved toward more practical concepts, so did the experimental designs, and two designs representing different compromises were developed [14, 296].

In the fall of 1983 an ambitious and successful VLSI project was undertaken by three students, Ackerman, Bauman and Woodhall. They designed a single chip embodying the control logic of a cell block together with a simple eight-bit arithmetic/logic unit. This chip was eventually refined, fabricated, and successfully tested in 1985 by Ackerman.

11 Routing Networks

Arbitration and distribution networks were integral parts of our early design proposals for data flow computers [129, 130]. Theoretical studies and simulations of these networks were carried out by Boughton [29], by Jacobsen and Misunas [189, 190], and in a bachelor's thesis [235, 234].

Once the cell block archictecture became the focus of our research, work on processor interconnection concentrated on packet-switched networks having a uniform data path width. On this basis, most of the conceptual difference between arbitration networks and distribution networks disappears, and a single routing module can serve as a building block for arbitrarily large machines.

Our first design of a two-by-two router module was done by Tai-Lai Tung, an undergraduate student technician. His successful design using asynchronous logic principles was validated and refined by Redford [276] and Lilienkamp [218, 219]. For use in the engineering model, diagnostic facilities were added to the design by Lim [220].

Subsequently, a VLSI implementation of the two-by-two packet router was proposed [279], and a very elegant self-timed realization was developed by Tam-Anh Chu [58], who also had the device fabricated through the MOSIS facility and verified correct operation of the product.

A general treatment of packet-routing principles for interconnecting large numbers of processing elements was completed in the doctoral research of Boughton [28, 30].

12 Fault Tolerance

The possibility of making data flow computers fault-tolerant has been a matter of interest to us since our earliest conception of data flow designs, as in the study by Misunas [246, 245]. The most extensive work was done more recently by Leung, who developed a general approach to making packet communication systems tolerant of arbitrary single fault occurrences [212, 213, 214, 215].

13 Functional Programming Languages: Val

Throughout the research program, we have always envisioned a textual language as the user language for data flow computers. Yet, it has been our philosophy to resolve semantic issues before dealing with their syntactic forms of expression.

We familiarized ourselves with modern ideas about formalisms for expressing the semantics of programming languages and systems, and with related work on structured programming, formal verification, and abstract data types. We perceived the importance of this work in providing criteria for distinguishing good programming constructs from bad ones: "goto considered harmful"; "side effects make program proofs difficult". We applied these perceptions in choosing the foundation constructs for data flow programming.

In 1978 our work with the Lawrence Livermore National Laboratory led to the definition of a user programming language for scientific data flow computation. Two pieces of background experience made this an easy task. One was the experience of Professor Dennis in teaching a course (MIT subject 6.534 "Semantic Theory for Computer Systems" beginning in spring 1973) in which the semantic notation of Scott was introduced in 1975 and used to give formal definition of a simple functional language. The second was Dennis' participation in the design process for the object-oriented language Clu conducted by Professor Liskov and her students [223]. In fact, the initial draft of the "Preliminary Manual for the Programming Language Val" [10] was created by editing the Clu programmer's manual. The utility of Val for research in compiling and programming parallel computation was promoted by James McGraw [231], and the unique features of Val relating to error-handling are presented in [315]. More recently, a structured editor for Val has been designed [229, 230].

The programming language SISAL [232] is derived from Val, and other workers have

independently designed functional languages for data flow computation, notably Id by Professor Arvind's MIT group [16], and Lapse by the Manchester University group.

At the time Val was defined, the design goal was to provide a basis for studying the scientific application of data flow computation. Nevertheless, much progress had been made toward the long range objective of a general-purpose language based on functional programming principles. Issues of free variables and their binding were resolved through a succession of analyses [86, 12, 13, 156]. Ackerman published an exposition on the change of thinking required when dealing with data arrays in the functional programming style [5, 6, 1].

Many authors have expressed dismay at the prospect of debugging large programs constructed for execution on highly parallel machines. One can sympathize with this view when the parallel machines are arrays of von Neumann processors and the programmer must express the coordination of actions using "synchronizing primitives" or "message passing" facilities outside the principal programming language (usually Fortran). On the contrary, use of a functional programming language implemented on a data flow machine makes possible a debugging support environment more attractive than that offered for conventional computers. A discussion of issues and a proposal are given by Bauman [21, 22]. Our research also includes an earlier study of debugging systems [208].

14 Applications of Data Flow Architecture

The study of specific applications for data flow computers has provided the means for a continuing evaluation of our architectural concepts. In each case, our study of an application has included hand construction of the graph of data flow instructions needed to implement critical parts of the algorithm. In each case, the experiment has yielded insights that led to improvements in our designs. One of the earliest subjects of study is the fast Fourier transform, for which data flow machine code was designed for an early version of the static data flow architecture [131, 126, 54, 55]. Other early studies concerned the problem of airplane collision avoidance [66, 162] and musical sound synthesis [51, 52], illustrating the broad range of application for data flow computation.

In many ways, the problems of modeling the global atmosphere is representative of large-scale scientific computation: the problem is reduced to solving a system of difference equations over a uniform, three-dimensional, rectangular grid. This class of computations has served as a principal guide in evaluating useful directions of evolution for data flow architecture. We began this work based on computational procedures used in the MIT Earth Sciences department [149, 254, 255, 138], and continued it later using a benchmark code supplied by the NASA Ames Research Center [122, 123].

Our collaboration with the Lawrence Livermore National Laboratory led to study of a benchmark hydrodynamics code known as Simple [253]. Although cut down from a classified

production weapons code, this program embodies the aspects of the application known to be particularly challenging for current supercomputers. Our analysis shows there is much concurrency available, but one part of the code requires a minor revision to remove a serious bottleneck, and use of a more parallel solver for the block tridiagonal linear systems arising in the computation would increase greatly the number of processors that could be used. The Simple code has been translated into Val and used for testing new compilation techniques for data flow computers.

The third substantial code studied is a benchmark program for three dimensional aerodynamic simulation. This was rewritten in Val and used as a tool for evaluating proposed data flow machine designs. The results of this study were reported to NASA [113, 114]. A recent study [79] presents a thorough analysis of this application based on the latest design proposals for data flow machines.

In many problems that involve solution of partial differential equations, the solution of linear equation systems having "tridiagonal" form is required. The problems of implementing "pipelined" data flow machine code for this important component of many scientific computations have been analyzed by Gao [155].

15 Program Structure and Compiler Technology

Our work with various application codes guided our study of machine code structures suitable for achieving high efficiency in static data flow computation. Our early efforts in this area led us to devote one of our summer workshop meetings to a review of contemporary projects [247].

A paper by Misunas [244] was an early contribution to the topic. Subsequent studies concerned buffering requirements [184], and economy in the use of acknowledgement arcs [249, 250]. Todd conceived elegant data flow machine code structures to implement the shared use of function bodies [299], and also gave the basic structures for the pipelined implementation of the Val **forall** and **for...iter** constructs [297, 300]. The latter work was subsequently expanded and refined by Gao Guang-Rong [153].

Only gradually did we come to understand the requirements to be satisfied by a compiler for data flow computers. One of the first issues to be clarified was the need to balance data flow graphs so they could be operated in pipelined mode. This was accomplished in the master's theses of Montz and Gao [249, 250, 153, 154]. Our knowledge increased as we developed machine features and code structures that would yield high utilization of the processing and memory resources of the machine. The problem of translating from a functional programming language into data flow graphs was formalized and given a careful treatment by Brock [37, 38], using ideas from semantic theory and from Weng's work [312].

The problem of generating efficient code for a data flow machine involves choosing an

optimal partitioning of the computation in space and time. For many scientific applications, the computation follows a sufficiently regular pattern that static analysis by a compiler can provide a good solution. Two approaches to this problem have been developed as the subject of doctoral research. In his thesis, William Ackerman makes use of the unrolling of iterative loops and the interleaving of array references to transform blocks of code so they may make use of many processing elements [8, 9]. In his thesis, Gao Guang-Rong applies transformations methodically so that each block of code becomes an efficient data flow pipeline [120, 121, 151, 152]. We expect that an effective compiler for a highly parallel computer will make use of both approaches.

We used some of the application codes as test vehicles for our compiling principles. For the weather code [123] and for the aerodynamic simulation problem [79], we constructed the data flow machine code that we envision a good compiler would produce.

16 Programming Generality

The application of data flow principles to large-scale scientific computation has received the greatest attention in our research program because this has been the most promising area for profitable near-term application. Nevertheless, the group has long been convinced of the applicability of data flow concepts to computing and information processing, generally. Indeed, Malhotra and Patil regarded their early work [226, 260] as new paradigms of computing that would be applicable to all domains of information processing. The philosophy of the group was expressed in a paper for the IFIP Congress 1968: "Programming Generality, Parallelism and Computer Architecture" [84]. The goal set was to achieve reusable software modules in the framework of a system in which all user information is held online in a hierarchical memory. One thesis put forth is that practical achievement of this goal would require use of parallel processing at fine grain level and use of a small unit of information transfer among the levels of the memory hierarchy. Our early work includes [23].

Some systems partially achieve this goal. The Multics operating system [64, 63] is an important milestone. Much later, the IBM System 38 incorporated many related ideas, although at considerable cost in performance and complexity of implementation.

An essay published in 1970 analyzes the issues involved in moving programs and data from one computing environment to another [76, 77]. The paper argues that the general problem of data exchange is no less difficult than the problem of program exchange, and therefore the concept of a "data description language" is not a solution.

In the Vim project, begun in 1981, the Computation Structures Group sought to test the validity of its concepts regarding the abstract computational model that would form the basis of a general-purpose, multiuser computer system based on principles of data flow computation and functional programming. The groundwork for such systems had already been laid in the doctoral theses of Rumbaugh [284, 285], Gertz [157, 158], Ellis [140, 141], Vanderbilt [304] and

Isaman [186, 187]. In 1979 Kung-Song Weng [313, 314] synthesized this earlier work into a comprehensive proposal for a computer system structure that would realize the goal. However, being a radically different structure from conventional computer systems, there is no reasonable way of evaluating its effectiveness other than by building it and trying it out. Yet, in building an unconventional system it is hard to predict the programming style that will be adopted by its users. Their programming style will certainly be different from that evolved for conventional computers, and it will likely be different from what the designers can envision.

For this reason, the Vim Project has the goal of building an implementation, functionally identical to the envisioned general-purpose dataflow machine, but built with commercially available components to reduce the design effort. A succession of operational models [110, 111, 164, 165, 191, 192] specify the *base language* for Vim. Dennis has described its conceptual basis [112] and the group has presented the planned internal mechanisms of the projected implementation [133, 134]. For the Vim Project, the Val language has been extended to incorporate high-order functions, streams, and a guardian construct to permit the programming of non-determinate computations [110]. At the end of funding, an implementation had been designed incorporating a compiler, a type-inference facility [266, 265, 206, 207], a scheme for data structure representation and implementation [164, 165], and a backup and recovery scheme [191, 192].

17 Petri Nets

Petri nets [267], originated by Carl Adam Petri [268] and introduced to American workers by Anatol Holt [183], are a scheme of representation for asychronous systems. They are attractive for formulating descriptions of logic systems, concurrent programs, systems of human interaction, etc. Largely at the instigation of Suhas Patil, the Computation Strucutures Group made many contributions to the theory and application of Petri nets.

Michel Hack made the principal theoretical contributions. He contributed to the classification of Petri nets and elucidated their properties [168, 169, 170]. He explored the relation of Petri nets to "vector addition systems" [171] and to formal languages [172]. He contributed to the resolution of the "reachability question" for Petri nets by providing insight to related questions of decidability for Petri nets [166, 167].

Petri net languages were also studied by Baker [20], and nets were used as a research tool for other studies in the group [177]. Furtek's work [150] elucidated the elegant structures and reasoning about concurrent sytems that may be mediated by the discipline of Petri nets.

A basic reason for our interest in Petri nets was their potential application to the design of asynchronous and self-timed systems. An early illustration of the power of these descriptive techniques was Dennis' description [85] of the control logic of the Control Data 6600 computer, the first commercially successful machine embodying instruction look-ahead and interleaved execution.

Some foundation principles for building logic circuits specified by Petri net descriptions were presented by Patil [262]. A good example of the design style developed by the group is Misunas' description of a self-timed realization of a rudimentary data flow computer [242]. Ramchandani introduced and developed the application of Petri nets to the performance analysis of systems encompassing many concurrent activities. For this work he introduced *timed Petri nets* in which the transitions have specified time delays [273, 274, 275], an idea also studied by Rotenberg [282]. Dennis developed tutorial material to show how Petri nets may be used to analyze software systems using concurrent processes [90, 96].

18 Self-Timed Systems

The early work on self-timed (or "speed-independent") logic circuits was done by David Muller at the Aiken Computation Laboratory of Harvard University [252], and at the University of Illinois in connection with construction of the Illiac II computer [251]. Since then, self-timed logic circuits have intrigued people. The possibility of building systems with distributed components but without a central source of timing signals was very appealing to us, and Muller's theory suggested that such systems could be built which would function correctly regardless of delays introduced by the active elements and interconnecting wires. Our studies led to a conception of digital systems as comprising a *data flow structure* built of registers and combinational logic blocks, and a *control structure* built of self-timed control modules. The power of this concept was shown in its application to describing the operation of the CDC 6600 processor [85]. Several other works of a similar nature were published [132, 264, 242, 159]. Subsequently Narinder Singh developed a design methodology based on the earlier foundation studies [286, 287].

Suhas Patil developed the idea of using Petri nets as a specification language for digital systems. He adopted for his doctoral research the problem of effectively implementing a useful class of Petri nets ("coordination nets") in digital logic [262, 257]. His research on this topic led to invention of the asynchronous logic array [263], which became the basis for a commercial enterprise.

One notorious issue for computer designers concerns synchronizers, devices that generate a pulse at the clock cycle next following the occurrence of an asynchronous event, typically a signal from an input or output device connected to a clocked computing system. It had long been known that such synchronizers will sometimes fail to perform correctly when the asynchronous event occurs within a critical interval of the clock transition. Our group has shown [239, 269] how asynchronous (self-timed) systems can be designed using asynchronous arbiters so that the function of synchronizing is performed, but without the hazard of incorrect operation always present in clocked systems using synchronizers. Patil showed that a bounded-delay arbiter and a synchronizer are equivalent [258, 259]. He also argued that perfect synchronizers and bounded-delay arbiters cannot be built, whereas an arbiter that has no time bound on its response to input signals can be built and will operate perfectly.

In 1980 we recognized that the advent of VLSI technology yielding high-density circuits might change the relative merits of clocked and self-timed design. Our summer workshop that year was devoted to a review of old and new ideas on the subject and an evaluation of their applicability in the VLSI era [46].

Recently our investigations have born fruit in the work of Tam-Anh Chu [56, 57, 58]. He has discovered a powerful new methodology for designing self-timed logic. Application of his concepts together with strategic exploitation of the layout possibilities offered by VLSI technology has produced several very effective designs: a self-timed router device [59] that could be used to build arbitrarily large packet-switched networks; and a buffer element [60] that could be used in the construction of an even more efficient router device. These designs have been fabricated and tested, and show that self-timed designs can be realized with no more logic complexity or silicon area than conventional clocked designs.

19 Semantic Theory for Functional Language and Architecture

In his opening remarks to the 1977 IFIP Working Conference on Formal Description of Programming Concepts [102], Professor Dennis emphasized the potential role of semantic theory in establishing the ideal form (base language) for computer systems. Semantic theory is useful not only for the precise and complete definition of programming languages, but also for defining the behavior of computer systems. At one level, a semantic description may define the "language" the system presents to the user, meaning those facilities offered by the operating system and the hardware as well as those offered by the user's chosen programming language. Here it is understandability that is most important—the user must readily comprehend system features so that they may be put to effective use. At another level, a formal description may represent the detailed actions by the elements that make up the system's implementation. Here what is important is the ability to ascertain consistency of the description with the designs of the elements used.

Our research group has been concerned with formal description techniques appropriate to these two levels of description, and with the methodology of establishing correctness (that is, consistency of the implementation with the semantics of the user "language"). Our approach to semantic specification was inspired by the work at the IBM Vienna Laboratory [224]. The doctoral research of Henderson [178] is one of the earliest uses of sophisticated operational semantic models in studying issues in computer system design. A pioneering effort in this area was Rumbaugh's doctoral thesis [284, 285] in which he formalized his original data flow architecture at the two levels mentioned, and presented the mappings between the two formal models that establish correctness of his proposed system design. Miranker [238] used similar techniques to establish formally the validity of his mechanisms for data flow procedure implementation.

The thesis of Ellis [140, 141] is a deeper study of a narrower domain, that of packet

communication systems. He gives a formal method of specifying determinate systems that accept and send out streams of items (possibly unending), and shows how to establish formally that an interconnection of such systems satisfies its specification.

The development of an adequate formal semantics for data flow programs was another area of interest. Dean Brock provided a consistent operational and denotational semantics for data flow program graphs and their corresponding expressions in the Val language [31, 32]. This work left open a crucial subject in semantic theory: the means of representing the behavior of nondeterminate actors (the nondeterminate merge actor) and programs containing them. The early proposals coming from the semantic theory community for mending the deficiency were not appealing to us because they lacked the property of being "fully abstract"—there was no composition rule that defined the semantics of a composite system in terms of the semantics of its components in a manner consistent with the operational semantics of data flow graphs. We wrestled with this problem for some time and developed some partial solutions [202, 199, 200, 201, 203]. Finally, the concept of *scenarios* was discovered by Brock and Ackerman [35, 36] and was refined into a theory of nondeterminate data flow programs [33, 34, 115].

A comprehensive bibliography of early literature on semantic theory was assembled by Steve Zilles [317].

20 Simulation

Simulation serves several roles in the study of computer system architecture. It can be used to test validity of an architectural concept. It can be used to estimate performance of a computer system or subsystem for a hypothesized load. It can be used to validate the functionality of a detailed design expressed as a logic design, a silicon layout, microcode, or register transfer description. Our research has used simulation in all of these ways.

The systems under study are themselves distributed systems of a special sort—packet communication systems. Randal Bryant has made contributions both to techniques for simulating packet communication systems [39, 40], and to carrying out simulation on distributed, multiprocessor computer systems [41, 42]. Simulation of our engineering model processing unit at the instruction set level was carried out by Resnick [277].

An early study of efficient digital logic simulation was done by Donald Smith [292, 293], and Randal Bryant introduced a very popular technology by developing switch-level simulation for VLSI circuits [43, 44, 45].

21 General

An important topic given insufficient attention by computer scientists is data bases. Since any practical computer system must provide support for data base storage and retrieval, we have been careful to ensure that our concepts of systems based on data flow principles are adequate to provide effective support for data base operations. The thesis research of Hawryskiewicz [124, 173, 174] showed how a relational data base model supporting multiple users and handling updates can be modeled on a base language substrate. In his doctoral thesis, Sheldon Borkin studied equivalence properties of the relational and semantic graph data models [25, 24, 26, 27].

Additional contributions by the research group include a review chapter on concurrent programming [48, 49, 47], a thesis comparing inductive program proof methods [160, 161], a primer for the Lisp machine [163], an example of programming with abstract data types [98], and a study of surveillance mechanisms [283]. The group has published a commentary on research directions in computer architecture [118], and a lecture series on data flow models of computation [115].

22 Conclusion

The effort reviewed here has been gratifying to the principal investigator. The goal of bringing new ideas from the germ of thought to practical realization and application has been nearly achieved. Many research institutions have taken up study of variants inspired by work on data flow computation done at MIT, several companies have taken steps toward familiarization with the ideas, and commercial products have been anounced by Nippon Electric and by Loral Instrumentation. Data flow technology has been under consideration for several years for certain military uses.

We appreciate the support provided by the sponsoring agencies during the period of development, and we are confident that the future will see their investment amply rewarded.

23 Personnel

The personnel who worked with Professor Dennis on the data flow project are listed below with the degrees awarded on the basis of research done in the Computation Structures Group. Students marked with an asterisk (*) are currently enrolled in degree programs.

Doctoral Students

William B. Ackerman	S.M. '77, Ph.D. '84
Sheldon A. Borkin	S.M. '78, Ph.D. '79
George Andrew Boughton	S.M. '78, Ph.D. '85
Jarvis Dean Brock	S.M. '79, Ph.D. '83
Randal Everitt Bryant	S.M. '77, Ph.D. '81
Tam-Anh Chu*	S.M. '82, Ph.D.'87
Peter James Denning	S.M. '65, Ph.D. '68
David J. Ellis	Ph.D. '74
Frederick C. Furtek	Ph.D. '76
Guang-Rong Gao	S.M. '82, Ph.D. '86
Jeffrey Lee Gertz	Ph.D. '70
Bhaskar Guharoy*	S.M. '85
Michel Henri Theodore Hack	S.M. '72, Ph.D. '75
Igor Titus Hawryskiewicz	Ph.D. '73
Prakash G. Hebalkar	S.M. '68, Ph.D. '70
D. Austin Henderson, Jr.	Ph.D. '75
David Lee Isaman	Ph.D. '79
Suresh Jagannathan*	S.M. '85
Paul Roman Kosinski	Ph.D '79
Clement Kin Cho Leung	S.M. '75, Ph.D. '81
John P. Linderman	Ph.D. '73
Fred Louis Luconi	Ph.D. '68
Suhas S. Patil	S.M. '67, Ph.D. '70
Joseph E. Qualitz	S.M. '72, Ph.D. '75
Chander Ramchandani	Ph.D. '74
Jorge E. Rodriguez	Ph.D. '67
James Rumbaugh	Ph.D. '75
Lawrence Seligman	
Donald Ray Slutz	Ph.D. '68
Earl Cornelius van Horn, Jr.	Ph.D. '66
Dean Hanawalt Vanderbilt	Ph.D. '69
Earl D. Waldin*	
Kung-Song Weng	S.M. '75, Ph.D. '79

Master's Degree Students

Surendra Nimal Amerasinghe	S.M. '72
Katsuhiko Amikura	S.M. '77
Donald J. Aoki	S.M. '79
Nena B. Bauman	S.M. '84

Peter B. Bishop	S.M. '72
Ian Richard Campbell-Grant	S.M. '71
Carol Andrea Cesari	S.M. '81
John Blake Fosseen	S.M. '72
Paul Jeffrey Fox	S.M. '73
Thomas McDonough Gearing	S.M. '73
Irene Gloria Greif	S.M. '72
Thomas R. Hegg	S.M. '83
Bradley C. Kuszmaul	S.B. '84, S.M. '86
Bruce P. Lester	S.M. '71
James William Leth	S.M. '79
Jeffrey Bruce Lotspiech	S.M. '72
Glen Seth Miranker	S.M. '77
David P. Misunas	S.B. '73, S.M. '75
Lynn B. Montz	S.M. '80
Richard Ribak	S.M. '68
Leo Joseph Rotenberg	S.M. '66
Narinder Pal Singh	S.M. '81
Arthur Anshel Smith	S.M. '66
Donald Leigh Smith	S.M. '66
Kevin B. Theobald	
Kenneth Wayne Todd	S.M. '81
Thomas S. Wanuga	S.M. '85
Michael J. Yachtman	

Undergraduate Students

Jeslie R. Chermak	S.B. '76
Arnold Chien	S.B. '80
Timothy Richard Connelly	S.B. '66
Lawrence Crooks	S.B. '76
Gilbert Falk	S.B. '65
Arif M. Feridun	S.B. '78
Edward Flinker	S.B. '72
Thomas B. Freeman	S.B. '77
David Gero	S.B. '72
Cindy Gilbert	
Charles A. Goldman	S.B. '86
Edward H. Gornish	S.B. '86
Steven Grossman	S.B. '77
Walid M. Hamdy	
Howard Louis Helman	S.B. '65
James E. Holderle	S.B. '84
Robert G. Jacobsen	S.B. '78
Klunder, D.B.	S.B. '71
David Kravitz	S.B. '85
Dana Alfred Lasher	S.B. '65
Joel Lilienkamp	S.B. '80

Beng-Hong Lim	S.B. '86
Edward H. Lyons	S.B. '86
David M. Marcovitz	S.B. '86
Steven C. Markowitz	S.B. '84
Mary E. McNally	S.B. '78
Barbara J. Miglierina	S.B. '76
Martin George Morris	S.B. '72
David R. Nadler	S.B. '78
Timothy Peacock	S.B. '83
John L. Redford	S.B. '79
Paul E. Ressler	S.B. '79
Paul S. Ries	S.B. '80
Bernard Hugh Robinson	S.B. '75
Rex Carroll Ross	S.B. '66
Matthew S. Stern	S.B. '82
Raymond S. Tetrick	S.B. '79
William T.-C. Tsang	
Richard Tucker	S.B. '80
Tai-Lai Tung	
Ephraim M. Vishniac	S.B. '79
Michael Stephen Wolfberg	S.B. '63

Research Staff

William B. Ackerman
G. Andrew Boughton
Denis J. Kfoury
William Y.-P. Lim
David P. Misunas
John M. Myers
Andrij Neczwid
William W. Plummer
Edward S. Shaw
Pedro S. Thiagarajan

Support Staff

Marsha Baker
Ilene Klang
Anne Rubin
Pamela Sedell
Natalie F. Tarbet

Visitors

Gary Lindstrom
Robert McNaughton
Toshio Shimada

Bard Sorbye
Joseph E. Stoy
James H. Vellenga

24 Bibliography

1. Ackerman, W. B. Data flow languages. *IEEE Computer 15*, 2 (February 1982), 15-25.

2. Ackerman, W. B. *A Structure Memory for Data Flow Computers.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, September 1977.

3. Ackerman, W. B. A Structure Controller for Data Flow Computers. MIT/LCS/TR-156, Laboratory for Computer Science, MIT, Cambridge, MA 02139, January, 1978.

4. Ackerman, W. B. A structure processing facility for data flow computers. Proc. of 1978 Intern. Conf. on Parallel Processing, Institute of Electrical and Electronics Engineers, Piscataway, N. J., 08854, New York, NY, August, 1978, pp. 166-172.

5. Ackerman, W. B. Data Flow Languages. Computation Structures Group Memo 177, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1979.

6. Ackerman, W. B. Data flow languages. Proc. of the 1979 Nat. Comp. Conf., June, 1979, pp. 1087-1095.

7. Ackerman, W. B. Processing Unit Programming Manual. MIT/LCS/TR-192, Computation Structures Group, Laboratory for Computer Science, MIT, Cambridge, MA 02139, April, 1980.

8. Ackerman, W. B. Efficient Implementation of Applicative Languages. MIT/LCS/TR-323, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1984.

9. Ackerman, W. B. *Efficient Implementation of Applicative Languages.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, April 1984.

10. Ackerman, W. B. and Dennis, J. B. VAL—A Value-oriented Algorithmic Language: Preliminary Reference Manual. Report MIT/LCS/TR-218, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1979.

11. Adams, D. A. A Computation Model with Data Flow Sequencing. Technical Report CS 117, Computer Science Department, Stanford University, Stanford, CA, December, 1968.

12. Amerasinghe, S. N. *Handling of Procedure Variables in a Base Language.* Master's Th., Department of Electrical Engineering and Computer Science, MIT, Cambridge, Mass., September 1972.

13. Amerasinghe, S. N. and Henderson, D. A., Jr. A Contour Model Evaluator for Lambda-Calculus Expressions, Computation Structures Group Memo 74, Project MAC, MIT, Cambridge, MA 02139, February, 1972.

14. Amikura, K. A Logic Design for the Cell Block of a Data Flow Processor. MIT/LCS/TM-93, Laboratory for Computer Science, MIT, Cambridge, MA 02139, December, 1977.

15. Arvind and Brock, J. D. Streams and managers . Proc. of the Fourteenth IBM Computer Science Symposium, June, 1982. Superseded by Arvind and Brock, J. D., Resource Managers in Functional Programming. *J. of Parallel and Distributed Computing 1*, (1984), 5-21.

16. Arvind, Gostelow, K., and Plouffe, W. The (Preliminary) Id Report: An Asynchronous Programming Language and Computing Machine. Technical Report 114, Department of Information and Computer Science, University of California, Irvine, September, 1978.

17. Arvind, Kathail, V. and Pingali, K. A Dataflow Architecture with Tagged Tokens. MIT/LCS/TM-174, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1980.

18. Arvind and Kathail, V. A multiple processor dataflow machine that supports generalized procedures. Proc. of the Eighth Ann. Symp. on Computer Architecture, May, 1981, pp. 291-296.

19. Arvind, and Kathail, V. A Multiple Processor Dataflow Machine that Supports Generalized Procedures. Computation Structures Group Memo 205-1, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1981.

20. Baker, H. Petri Nets and Languages. Computation Structures Group Memo 68, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1972.

21. Bauman, N. *A Debugging Process for Static Programs.* Master's Th., Department of Computer Science and Electrical Engineering, MIT, Cambridge, Mass., September 1984.

22. Bauman, N. and Iannucci, R. A. A Methodology for Debugging Data Flow Programs. MIT/LCS/TR-219, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1982.

23. Bishop, P. B. *Data Types for Programming Generality.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1972.

24. Borkin, S. Data model equivalence. Proc. of the Fourth Intern. Conf. on Very Large Data Bases, IEEE, New York, NY, September, 1978, pp. 526-534.

25. Borkin, S. Data Model Equivalence. Computation Structures Group Memo 217, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1978.

26. Borkin, S. Equivalence Properties of Semantic Data Models for Data Bases. MIT/LCS/TR-206, Laboratory for Computer Science, MIT, Cambridge, MA 02139, January, 1979.

27. Borkin, S. *Equivalence Properties of Semantic Data Models for Data Bases.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1979.

28. Boughton, G. A. Routing Networks for Packet Communication Systems. MIT/LCS/TR-341, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1985.

29. Boughton, G. A. *Routing Networks in Packet Communication Architectures.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1978.

30. Boughton, G. A. *Routing Networks for Packet Communication Systems.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, September 1985.

31. Brock, J. D. Consistent semantics for a dataflow language. In *Lecture Notes in Computer Science, Volume 88: Mathematical Foundations of Computer Science.* Springer-Verlag, Berlin, Heidelberg, New York, 1980, pp. 168-180.

32. Brock, J. D. Consistent Semantics for a Dataflow Language. Computation Structures Group Memo 172, Laboratory for Computer Science, MIT, Cambridge, MA 02139, January, 1979.

33. Brock, J. D. *A Formal Model of Non-determinate Data Flow Computation.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, August 1984.

34. Brock, J. D. A Formal Model of Non-determinate Data Flow Computation. MIT/LCS/TR-309, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1984.

35. Brock, J. D. and Ackerman, W. B. Scenarios: A Model of Non-determinate Computation. Computation Structures Group Memo 206, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1981.

36. Brock, J. D. and Ackerman, W. B. Scenarios: a model of non-determinate computation. In *Lecture Notes in Computer Science, Volume 107: Formalization of Mathematical Concepts.* Springer-Verlag, Berlin, Heidelberg, New York, 1981, pp. 252-259.

37. Brock, J. D. and Montz, L. Translation and optimization of data flow programs. Proc. of the 1979 Intern. Conf. on Parallel Processing, IEEE, New York, NY, August, 1979, pp. 46-54.

38. Brock, J. D. and Montz, L. Translation and Optimization of Data Flow Programs. Computation Structures Group Memo 181, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1979.

39. Bryant, R. E. Simulation of Packet Communication Architecture Computer Systems. MIT/LCS/TR-188, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1977.

40. Bryant, R. E. *Simulation of Packet Communication Architecture Computer Systems.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, November 1977.

41. Bryant, R. E. Simulation on a distributed system. Proc. of the First Intern. Conf. on Distributed Computing Systems, IEEE, New York, NY, October, 1979, pp. 544-552.

42. Bryant, R. E. Simulation on a Distributed System. Computation Structures Group Memo 182, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1979.

43. Bryant, R. E. An algorithm for MOS logic simulation. *Lambda Magazine I* (Fourth Quarter 1980), 46-53.

44. Bryant, R. E. A Switch-level Simulation Model for Integrated Logic Circuits. MIT/LCS/TR-259, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1981.

45. Bryant, R. E. *A Switch-level Simulation Model for Integrated Logic Circuits.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, March 1981.

46. Bryant, R. E. Report on the Workshop for Self-timed Systems. MIT/LCS/TM-166, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1981.

47. Bryant, R. E. and Dennis, J. B. Concurrent Programming. In *Research Directions in Software Technology.* MIT Press, Cambridge, MA 02139, 1978, pp. 584-610.

48. Bryant, R. E. and Dennis, J. B. Concurrent Programming. Computation Structures Group Memo 148-2, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1978.

49. Bryant, R. E. and Dennis, J. B. Concurrent Programming. MIT/LCS/TM-115, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1978.

50. Campbell-Grant, I. *The Controlled Execution of Parallel Programs Operating on Structured Data.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1971.

51. Cesari, C. A. Applications of a Data Flow Architecture to Computer Music Synthesis. MIT/LCS/TR-257, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1981.

52. Cesari, C. A. *Applications of a Data Flow Architecture to Computer Music Synthesis.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1981.

53. Chermak, J. C. Packet Communication Discipline in Microprocessor, Microprocessing, Packet-driven System. Unpublished S.B. thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139.

54. Chien, A. Structuring the Fast Fourier Transform for Data Flow Computation. Computation Structures Group Memo 193, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1980.

55. Chien, A. Structuring the Fast Fourier Transform for Dataflow Computation. S.B. thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139.

56. Chu, T.-A. A Design Strategy for Testable Self-timed Systems. Computation Structures Group Memo 216, Laboratory for Computer Science, MIT, Cambridge, MA 02139, April, 1982.

57. Chu, T.-A. Circuit Analysis of Self-timed Elements for NMOS VLSI Systems. MIT/LCS/TM-220, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1982.

58. Chu, T.-A. The Design, Implementation and Testing of a Self-timed Two-by-Two Packet Router. Computation Structures Group Memo 225, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1983.

59. Chu, T.-A. Design of a CMOS Self-timed Two-by-Two Packet Router. Computation Structures Group Memo 242, Laboratory for Computer Science, MIT, Cambridge, MA 02139, December, 1984.

60. Chu, T.-A. Design of a VLSI Self-timed Ring Buffer Using Signal Transition Graphs. Computation Structures Group Memo 247, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1985.

61. Conway, M. E. A multiprocessor system design. AFIPS Conference Proceedings, 1963, pp. 139-146.

62. Conway, M. E. Design of a separable transition-diagram compiler. *Communications of the ACM 6*, 7 (July 1963), 396-408.

63. Corbato, F. J. Multics—the first seven years. AFIPS Conference Proceedings, 1972, pp. 571-583.

64. Corbato, F. J., and Vyssotsky, V. A. Introduction and Overview of the Multics System. AFIPS Conference Proceedings, 1965, pp. 185-196.

65. Cornish, M. The TI dataflow architecture: the power of concurrency for avionics. Proc. of the Third Digital Avionics Systems Conf., 1979, pp. 19-25.

66. Crooks, L. Analysis of Airplane Collision Avoidance Algorithm. Unpublished S.B. thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139.

67. Daley, R. C. and Dennis, J. B. Virtual memory, processes and sharing Multics. Proc. of the Sym. on Operating System Principles, October, 1967.

68. Daley, R. C. and Dennis, J. B. Virtual memory, processes and sharing in Multics. *Communications of the ACM 11*, 5 (May 1968), 306-312.

69. Denning, P. J. On the determinacy of schemata. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, New York, NY, 1970, pp. 143-147.

70. Denning, P. J. Queueing Models for File Memory Operation. MIT/LCS/TR-21, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1965.

71. Denning, P. J. The working set model for program behavior. Proc. of the Symp. on Operating System Principles, October, 1967.

72. Denning, P. J. Effects of scheduling on file memory operations. Proc. of the Spring Joint Computer Conf., AFIPS, Washington, DC, 1967, pp. 9-21.

73. Denning, P. J. The working set model for program behavior. *Communications of the ACM 11*, 5 (May 1968), 323-333.

74. Denning, P. J. Resource Allocation in Multiprocess Computer Systems. MIT/LCS/TR-50, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1968.

75. Denning, P. J. *Resource Allocation in Multiprocess Computer Systems.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, May 1968.

76. Dennis, J. B. On the exchange of information. Computation Structures Group Memo 52, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1970.

77. Dennis, J. B. On the exchange of information. Proceeding of ACM-SIGFIDET Workshop on Data Description and Access, New York, NY, 1970.

78. Dennis, J. B. Data flow ideas for supercomputers. Proc. of the Twenty-eighth IEEE Computer Society Conf., February, 1984, pp. 15-19.

79. Dennis, J. B. Dataflow computation: a case study. In *Computer Architecture: Concepts and Systems.* V. Milutinovic, Ed., Elsevier, New York, 1987.

80. Dennis, J. B. Program Structure in a Multi-access Computer. MIT/LCS/TR-11, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1964.

81. Dennis, J. B. Segmentation and the design of multiprogrammed computer systems. *Journal of the ACM 8*, 10 (October 1965), 589-602.

82. Dennis, J. B. A position paper on computing and communications. Proc. of the Symp. on Operating System Principles, ACM, October, 1967.

83. Dennis, J. B. A position paper on computing and communications. *Communications of the ACM 11*, 5 (May 1968), 370-377.

84. Dennis, J. B. Programming generality, parallelism and computer architecture. In *Information Processing 68.* North-Holland, Amsterdam, 1969, pp. 484-492.

85. 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, pp. 55-80.

86. Dennis, J. B. On the design and specification of a common base language. Proc. of the Symposium on Computers and Automata, Brooklyn, NY, 1971, pp. 47-74.

87. Dennis, J. B. Coroutines and parallel computation. Princeton Conference on Information Sciences and Systems, Princeton, NJ, March, 1971, pp. 293-294.

88. Dennis, J. B. Management of Names in a Computer System. Computation Structures Group Memo 63, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1971.

89. Dennis, J. B. Design and Construction of Software Systems. Computation Structures Group Memo 69, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1972.

90. Dennis, J. B. Concurrency in Software Systems. Computation Structures Group Memo 65-1, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1972.

91. Dennis, J. B. Modularity. Computation Structures Group Memo 70, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1972.

92. Dennis, J. B. A multiuser computation facility for education and research. *Communications of the ACM 7*, 9 (September 1974), 521-529.

93. Dennis, J. B. First version of a data flow procedure language. In *Lecture Notes in Computer Science, Volume 19: Programming Symposium*. B. Robinet, Ed., Springer-Verlag, Berlin, Heidelberg, New York, 1974, pp. 362-376.

94. Dennis, J. B. On Storage Management for Advanced Programming Languages. Computation Structures Group Memo 109, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1974.

95. Dennis, J. B. Packet communication architecture. Proc. of the 1975 Sagamore Conf. on Parallel Processing, 1975, pp. 224-229.

96. Dennis, J. B. Modularity; Concurrency in software systems; and On the design and construction of software systems. In *Lecture Notes In Computer Science, Volume 30: Software Engineering: An Advanced Course*. Springer-Verlag, Berlin, Heidelberg, New York, 1975, pp. 12-28, 111-127, 128-182.

97. Dennis, J. B. Packet Communication Architecture. Computation Structures Group Memo 130, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1975.

98. Dennis, J. B. An Example of Programming with Abstract Data Types. Computation Structures Group Memo 131, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1975.

99. Dennis, J. B. Proposed Research on Architectural Principles for Large Memory Systems. Computation Structures Group Memo 132, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1975.

100. Dennis, J. B. Computer architecture and the cost of software. *ACM Computer Architecture News 4* (April 1976).

101. Dennis, J. B. Computer Architecture and the Cost of Software. Computation Structures Group Memo 140, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1976.

102. Dennis, J. B. Opening Remarks to the IFIP Working Conference on Formal Description of Programming Concepts (Saint Andrews, NB, Canada). Computation Structures Group Memo 152, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1977.

103. Dennis, J. B. A language design for structured concurrency. In *Lecture Notes in Computer Science, Volume 54: Design and Implementation of Programming Languages*. J. H. Williams and D. A. Fisher, Eds., Springer-Verlag, Berlin, Heidelberg, New York, 1977.

104. Dennis, J. B. Data Flow Computer Architecture. Computation Structures Group Memo 160, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1978. Research proposal submitted to the Department of Energy.

105. Dennis, J. B. Data Flow Computer Architecture. Computation Structures Group Memo 174, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1979. Research proposal submitted to the National Science Foundation.

106. Dennis, J. B. The varieties of data flow computers. Proc. of the First Intern. Conf. on Distributed Computing Systems, Huntsville, AL, October, 1979, pp. 430-439.

107. Dennis, J. B. The Varieties of Data Flow Computers. Computation Structures Group Memo 183-1, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1979.

108. Dennis, J. B. Data Flow Computer Architecture. Computation Structures Group Memo 198, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1980. Research proposal submitted to the Department of Energy.

109. Dennis, J. B. Data flow supercomputers. *IEEE Computer 13*, 11 (November 1980), 48-56.

110. Dennis, J. B. An operational semantics for a language with early completion data structures. In *Lecture Notes in Computer Science, Volume 107: Formal Description of Programming Concepts*. Springer-Verlag, Berlin, Heidelberg, New York, 1981, pp. 260-267.

111. Dennis, J. B. An Operational Semantics for a Language with Early Completion Data Structures. Computation Structures Group Memo 207, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1981.

112. Dennis, J. B. Data Should Not Change: A Model for a Computer System. Computation Structures Group Memo 209, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1981.

113. Dennis, J. B. . High Speed Data Flow Computer Architecture for the Solution of Navier-Stokes Equations. Laboratory for Computer Science, MIT, Cambridge, MA 02139, 1982.

114. Dennis, J. B. High Speed Data Flow Computer Architecture for the Solution of Navier-Stokes Equations. Computation Structures Group Memo 225, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1983.

115. Dennis, J. B. Data flow models of computation. In *Control Flow and Data Flow: Concepts of Distributed Programming*. M. Broy, Ed., Springer-Verlag, Berlin, Heidelberg, New York, 1984.

116. Dennis, J. B., Boughton, G. A., and Leung, C. Building blocks for data flow prototypes. Proc. of the Seventh Ann. Symp. on Computer Architecture, LaBaule, France, May, 1980, pp. 1-8.

117. Dennis, J. B., Boughton, G. A., and Leung, C. Building Blocks for Data Flow Prototypes. Computation Structures Group Memo 190, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1980.

118. Dennis, J. B. *et al.*. Research Directions in Computer Architecture. Computation Structures Group Memo 114, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1978.

119. Dennis, J. B., Fosseen, J. B. and Linderman, J. P. Data flow schemas. In *Lecture Notes in Computer Science, Volume 5: International Symposium on Theoretical Programming.* Springer-Verlag, Berlin, Heidelberg, New York, 1972, pp. 187-215. Also published in Russian.

120. Dennis, J. B. and Gao, G-R. Maximum Pipelining of Array Operations on Static Data Flow Machine. Computation Structures Group Memo 233, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1983.

121. Dennis, J. B. and Gao, G-R. Maximum pipelining of array operations on static data flow machine. Proc. of Intern. Conf. on Parallel Processing, August, 1983, pp. 331-335.

122. Dennis, J. B., Gao, G-R., and Todd, K. W. R. A Data Flow Supercomputer. Computation Structures Group Memo 213, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1982.

123. Dennis, J. B., Gao, G-R., and Todd, K. W. R. Modeling the weather with a data flow supercomputer. *IEEE Transactions on Computers C-33,7* (July 1984), 592-603.

124. Dennis, J. B. and Hawryszkiewicz, I. T. An approach to proving the correctness of data base operations. In *Workshop on Data Base Description, Access and Control.* ACM, 1973.

125. Dennis, J. B., Lim, W. L.-P., and Ackerman, W. B. The MIT data flow engineering model. Information Processing, IFIP, 1983, pp. 553-560.

126. Dennis, J. B., Leung, C. and Misunas, D. A Highly Parallel Computer Using a Data Flow Machine Language. Computation Structures Group Memo 134-1, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1979.

127. Dennis, J. B., Lim, W. L.-P., and Ackerman, W. B. The MIT Data Flow Engineering Model. Computation Structures Group Memo 222, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1982.

128. Dennis, J. B. and Misunas, D. P. A computer architecture for highly parallel signal processing. Proc. of the 1974 National Conf., ACM, 1974, pp. 402-409.

129. Dennis, J. B. and Misunas, D. P. A preliminary architecture for a basic data flow computer. Proc. of the Second Ann. Symp. on Computer Architecture, IEEE, New York, NY, 1975.

130. Dennis, J. B. and Misunas, D. P. A Preliminary Architecture for a Basic Data Flow Computer. Computation Structures Group Memo 102, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1975.

131. Dennis, J. B., Misunas, D. P., and Leung, C. A Highly Parallel Processor Based on the Data Flow Concept. Computation Structures Group Memo 134, Laboratory for Computer Science, MIT, Cambridge, MA 02139, January, 1976. Superceded by CSG Memo 134-1, June 1977, and CSG Memo 134-2, June 1980.

132. Dennis, J. B. and Patil, S. S. Speed independent asynchronous circuits. Proc. of the Fourth Hawaii Intern. Conf. on System Sciences, 1971, pp. 55-58.

133. Dennis, J. B., Stoy, J. E., and Guharoy, B. VIM: An Experimental Multi-User System Supporting Functional Programming. Computation Structures Group Memo 238, Laboratory for Computer Science, MIT, Cambridge, MA 02139, April, 1984.

134. Dennis, J. B., Stoy, J. E., and Guharoy, B. VIM: an experimental multi-user system supporting functional programming. Proc. of the Workshop on High-level Computer Architecture, May, 1984.

135. Dennis, J. B. and van Horn, E. Programming Semantics for Multiprogrammed Computations. MIT/LCS/TR-21, Laboratory for Computer Science, MIT, Cambridge, MA 02139, December, 1965.

136. Dennis, J. B. and van Horn, E. Programming semantics for multiprogrammed computations. *Communications of the ACM 9*, 3 (March 1966), 143-155.

137. Dennis, J. B. and Weng, K.-S. An abstract implementation for concurrent computation with streams. Proc. of the 1979 Conf. on Parallel Processing, August, 1979, pp. 35-45.

138. Dennis, J. B. and Weng, K.-S. Application of Data Flow Computation to the Weather Problem. Computation Structures Group Memo 147, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1977.

139. Ellis, D. J. Semantics of Data Structures. MIT/LCS/TR-134, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1974.

140. Ellis, D. J. Formal Specification for Packet Communication Systems. MIT/LCS/TR-189, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1977.

141. Ellis, D. J. *Formal Specification for Packet Communication Systems.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, November 1977.

142. Fabry, R. S. Capability-based addressing. *Communications of the ACM 17*, 7 (July 1974), 403-412.

143. Fano, R. M. The MAC system: the computer utility approach. *IEEE Spectrum 2*, 1 (January 1965).

144. Fano, R. M., and Corbato, F. J. Time-sharing on computers. *Scientific American 215*, 3 (September 1966), 128-140.

145. Feridun, A. Design of a Byte-level Pipelined Arithmetic Processor. Computation Structures Group Memo 162, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1978.

146. Feridun, A. M. *Design of an On-line Byte-level Pipelined Arithmetic Processor.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1978.

147. Fosseen, J. B. *Representation of Algorithms by Maximally Parallel Schemata.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1972.

148. Fox, P. J. *Representation of Parallel Computation on Data Structures.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1973. Also submitted for the E.E. degree, January 1973.

149. Freeman, T. B. *A Data Flow Program for Numerical Weather Prediction.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1977.

150. Furtek F. The Logic of Systems. MIT/LCS/TR-170, Laboratory for Computer Science, MIT, Cambridge, MA 02139, December, 1976.

151. Gao, G.-R. *A Pipelined Code Mapping Scheme for Static Data Flow Computers.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, August 1986.

152. Gao, G.-R. A Pipelined Code Mapping Scheme for Static Data Flow Computers. MIT/LCS/TR-371, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1986.

153. Gao, G-R. An Implementation Scheme for Array Operations in Static Data Flow Computers. MIT/LCS/TR-280, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1982.

154. Gao, G-R. *An Implementation Scheme for Array Operations in Static Data Flow Computers.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1982.

155. Gao, G-R. A Maximally Pipelined Tridiagonal Linear Equation Solver. Computation Structures Group Memo 254, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1985.

156. Gearing, T. M. *An Alternate Approach to the Furnag Problem in the Base Language.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1973.

157. Gertz, J. Hierarchical Associative Memories for Parallel Computation. MIT/LCS/TR-69, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1970.

158. Gertz, J. *Hierarchical Associative Memories for Parallel Computation.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1970.

159. Goldberg, H. J. An Asynchronous Model of a Small Computer. Computation Structures Group Memo 133, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1975.

160. Greif, I. Induction in Proofs About Programs. MIT/LCS/TR-93, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1972.

161. Greif, I. *Induction in Proofs About Programs.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1972.

162. Grossman, S. *A Dataflow Machine for an Airplane Collision Avoidance Network.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1977.

163. Guharoy, B. A Primer for the Lisp Machine. Computation Structures Group Memo 237, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1984.

164. Guharoy, B. *Data Structure Management in a Data Flow Computer System.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, May 1985.

165. Guharoy, B. Data Structure Management in a Data Flow Computer System. MIT/LCS/TR-355, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1985.

166. Hack, M. Decidability Questions for Petri Nets. MIT/LCS/TR-161, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1976.

167. Hack, M. *Decidability Questions for Petri Nets.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1976.

168. Hack, M. Analysis of Production Schemata by Petri Nets. MIT/LCS/TR-94, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1972.

169. Hack, M. *Analysis of Production Schemata by Petri Nets.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1972.

170. Hack, M. Extended State-Machine Allocatable Nets (ESMA), and Extension of Free Choice Petri Net Results. Computation Structures Group Memo 78, Project MAC, MIT, Cambridge, MA 02139, May, 1973.

171. Hack, M. Decision Problems for Petri Nets and Vector Addition Systems. MIT/LCS/TM-59, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1975.

172. Hack, M. Petri Net Languages. MIT/LCS/TR-159, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1976.

173. Hawryskiewicz, I. Semantics of Data Base Systems. MIT/LCS/TR-112, Laboratory for Computer Science, MIT, Cambridge, MA 02139, December, 1973.

174. Hawryskiewicz, I. *Semantics of Data Base Systems.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, December 1973.

175. Hebalkar, P. G. *Deadlock-free Sharing of Resources in Asynchronous Systems.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, September 1970.

176. Hebalkar, P. G. *Asynchronous Cooperative Multiprocessing within Multics.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1968.

177. Hebalkar, P. G. Deadlock-free Sharing of Resources in Asynchronous Systems. MIT/LCS/TR-75, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1970.

178. Henderson, D. A. The Binding Model: A Semantic Base for Modular Programming Systems. MIT/LCS/TR-145, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1975.

179. Hewitt, C. Viewing control structures as patterns of message passing. *Artificial Intelligence 8* (1977), 323-364.

180. Hoare, C. A. R. Monitors: an operating system structuring concept. *Communications of the ACM 17,* 10 (October 1975), 549-557.

181. Holderle, J. *A Parser for the Language PADL.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, August 1983.

182. Holderle, J. A Parser for the Language PADL. Computation Structures Group Memo 234, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1983.

183. Holt, A., and Commoner. F. Events and Conditions. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, Association for Computing Machinery, New York, NY, 1970, pp. 3-52.

184. Hong, P. *Analysis of Buffering Requirements in a Data Flow Processor.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1977.

185. IBM. *IBM System/38 Technical Developments.* IBM General Systems Division, 1978.

186. Isaman, D. Data-structuring Operations in Concurrent Computations. MIT/LCS/TR-224, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1979.

187. Isaman, D. *Data-structuring Operations in Concurrent Computations.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, October 1979.

188. Jacobsen, R. G. and Misunas, D. P. Analysis of structures for packet communication. Proc. of the 1977 Intern. Conf. on Parallel Processing, IEEE, New York, NY, August, 1977, pp. 38-43.

189. Jacobsen, R. G. *Analysis of Structures for Packet Sorting Networks.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1978.

190. Jacobsen, R. G. and Misunas, D. P. Analysis of Structures for Packet Communication. Computation Structures Group Memo 151, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1977.

191. Jagannathan, S. *Data Backup and Recovery in a Computer Architecture for Functional Programming.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, October 1985.

192. Jagannathan, S. Data Backup and Recovery in a Computer Architecture for Functional Programming. MIT/LCS/TR-353, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1985.

193. Johnson. D., et. al. Automatic Partitioning of Programs in Multiprocessor Systems. COMPCON Spring 80, Institute of Electrical and Electronics Engineers, Piscataway, N. J., 08854, February, 1980, pp. 175-178.

194. Kahn, G. The semantics of a simple language for parallel programming. In *Information Processing 74.* North Holland, Amsterdam, 1974, pp. 471-475.

195. Karp, R. M., and Miller, R. E. Properties of a model for parallel computations: determinacy, termination, queueing. *SIAM Journal of Applied Mathematics 14,* 6 (November 1966), 1390-1411".

196. Karp, R. M., and Miller, R. E. Parallel program schemata. *Journal of Computer and System Sciences 3*, 2 (May 1969), 147-195.

197. Keller, R., Lindstrom, G., and Patil, S. A loosely-coupled applicative multi-processing system. Proc. of the National Computer Conference, Association for Computing Machinery, June, 1979, pp. 613-622.

198. Kosinski, P. R. A Data Flow Programming Language. RC4264, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1973.

199. Kosinski, P. R. A Straightforward Denotational Semantics for Non-determinate Data Flow Programs. Computation Structures Group Memo 157, Laboratory for Computer Science, MIT, Cambridge, MA 02139, January, 1977.

200. Kosinski, P. R. A straightforward denotational semantics for non-determinate data flow programs. Proc. of the Fifth ACM Symposium on Principles of Programming Languages, January, 1978, pp. 214-219.

201. Kosinski, P. R. Denotational Semantics of Determinate and Non-determinate Data Flow Programs. MIT/LCS/TR-220, Laboratory for Computer Science, MIT, Cambridge, MA 02139, January, 1979.

202. Kosinski, P. R. Mathematical Semantics and Data Flow Programming. Computation Structures Group Memo 135, Laboratory for Computer Science, MIT, Cambridge, MA 02139, December, 1976.

203. Kosinski, P. R. *Denotational Semantics of Determinate and Non-determinate Data Flow.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1979.

204. Kung, H. T. Why systolic architecture? *IEEE Computer 16*, 1 (1982), 37-46.

205. Kurokawa, H., Matsumoto, K., Temma, T., Iwashita, M., and Nukiyama, T. The architecture and performance of image pipeline processor. Proc. of IFIP WG 10.5 Intern. Conference on Very Large Scale Integration, 1983.

206. Kuszmaul, B. C. Type-checking in VimVal. MIT/LCS/TR-309, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1984.

207. Kuszmaul, B.C. *Type-checking in VimVal.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1984.

208. Lester, B. Cost Analysis of Debugging Systems. MIT/LCS/TR-90, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1971.

209. Leung, C. K.-C. Formal Properties of Well-formed Data Flow Schemas. MIT/LCS/TM-66, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1975.

210. Leung, C. K.-C. *Formal Properties of Well-formed Data Flow Schemas.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1975.

211. Leung, C. K.-C. ADL: An Architecture Describing Language for Packet Communication Systems. Computation Structures Group Memo 185, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1979.

212. Leung, C. K.-C. Fault Tolerance in Packet Communication Computer Architectures. MIT/LCS/TR-250, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1980.

213. Leung, C. K.-C. *Fault Tolerance in Packet Communication Computer Architectures.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, September 1980.

214. Leung, C. K.-C. N. and Dennis, J. B. Design of a Fault-tolerant Packet Communication Computer Architecture. Computation Structures Group Memo 196, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1980.

215. Leung, C. K.-C. and Dennis, J. B. Design of a fault-tolerant packet communication computer architecture. Tenth Ann. Symp. on Fault Tolerant Computer Systems, IEEE, New York, NY, October, 1980, pp. 328-335.

216. Leung, C. and Lim, W. L.-P. A Packet Architecture Description Language. MIT/LCS/TR-306, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1982.

217. Leung, C., Misunas, D. P., Neczwid, A. R., and Dennis, J. B. A computer simulation facility for packet communication architecture. Proc. of the Third Ann. Symp. on Computer Architecture, IEEE, New York, NY, 1976, pp. 58-63.

218. Lilienkamp, J. The Development of a Prototype Router: Design, Implementation and Test Procedures. Computation Structures Group Memo 199, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1980.

219. Lilienkamp, J. *The Development of a Prototype Router: Design, Implementation and Test Procedures.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1980.

220. Lim, W. Y-P. Diagnostic Hardware of the Prototype 2 x 2 Router. Static Dataflow Machine Project Design Note 3, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1982.

221. Linderman, J. *Productivity in Parallel Computation Schemata.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, December 1973.

222. Linderman, J. Productivity in Parallel Computation Schemata. MIT/LCS/TR-111, Laboratory for Computer Science, MIT, Cambridge, MA 02139, December, 1973.

223. Liskov, B. H. et al. *Lecture Notes in Computer Science.* Volume 114: *CLU Reference Manual.* Springer-Verlag, Berlin, Heidelberg, New York, 1981.

224. Luconi, F. Asynchronous Computation Structures. MIT/LCS/TR-49, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1968.

225. Luconi, F. *Asynchronous Computation Structures.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1968.

226. Malhotra, A. *Asynchronous Control of Computer Operations.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1967.

227. Marcovitz, D. M. A Comparison of Two Signal System Architectures for a Static Dataflow Machine. Computation Structures Group Memo 260, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1986.

228. Marcovitz, D. M. *A Comparison of Two Signal System Architectures for a Static Dataflow Machine.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1986.

229. Markowitz, S. VLOE: A Val Language-oriented Editor. Computation Structures Group Memo 255, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1984.

230. Markowitz, S. *VLOE: A Val Language-oriented Editor.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1984.

231. McGraw, J. R. The VAL language: decription and analysis. *Transactions on Programming Languages and Systems 4*, 1 (January 1982), 44-82.

232. McGraw, J. R., Skedzielewski, S., Allan, S., Oldehoeft, R., Glauert, J., Kirkham, C., Noyce, W., and Thomas, R. SISAL: Streams and Iteration in a Single-Assignment Language. Language Reference Manual M-146, Lawrence Livermore Laboratory, Livermore, CA, March, 1985.

233. McIlroy, M. D. Coroutines: Semantics in Search of a Syntax. Unpublished memorandum, Oxford University and Bell Telephone Laboratories, 1968.

234. McNally, M. E. *The Design of an Arbitration Network for a Data Flow Processor.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1978.

235. McNally, M. E. The Design of an Arbitration Network for a Data Flow Processor. Computation Structures Group Memo 164, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1978.

236. Miglierina, B. J. *A Generalized Modular Microprocessor Simulator.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, 1976.

237. Miranker, G. S. Implementation Schemes for Data Flow Procedures. Computation Structures Group Memo 138, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1976.

238. Miranker, G. S. Proving Packet Communications Architectures Correct. Computation Structures Group Memo 143, Laboratory for Computer Science, MIT, Cambridge, MA 02139, 1976.

239. Miranker, G. S. The Synchronizer Problem. Computation Structures Group Memo 145, Laboratory for Computer Science, MIT, Cambridge, MA 02139, January, 1977.

240. Miranker, G. S. Implementation of procedures on a class of data flow processors. Proc. of the 1977 Intern. Conf. on Parallel Processing, IEEE, New York, NY, 1977, pp. 77-86.

241. Miranker, G. S. *Implementation Issues in Data Flow Architecture.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, 1977.

242. Misunas, D. P. Petri nets and speed independent design. *Communications of the ACM 16,* 8 (August 1973), 474-481.

243. Misunas, D. P. Deadlock avoidance in data flow architecture. Proc. of the Third Milwaukee Symp. on Automatic Computation and Control, April, 1975 .

244. Misunas, D. P. Structured programming in a data flow computer. Proc. of the 1975 Sagamore Conf. on Parallel Processing, IEEE, New York, NY, 1975, pp. 230-234.

245. Misunas, D. P. Error Detection and Recovery in a Data Flow Computer. Proc. of the Intern. Conf. on Parallel Processing, IEEE, New York, NY, 1976, pp. 117-122.

246. Misunas, D. P. Error Detection and Recovery in a Data Flow Computer. Computation Structures Group Memo 142, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1976.

247. Misunas, D. P. Report on the workshop on data flow computer and program organization. *ACM Computer Architecture News 5* (October 1977), 6-22.

248. Misunas, D. P. Report on the Workshop on Data Flow Computer and Program Organization. MIT/LCS/TM-92, Laboratory for Computer Science, MIT, Cambridge, MA 02139, November, 1977.

249. Montz, L. *Safety and Optimization Transformations for Data Flow Programs .* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, July 1980.

250. Montz, L. Safety and Optimization Transformations for Data Flow Programs. Computation Structures Group Memo 164, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1980.

251. Muller, D. E. Asynchronous logics and application to information processing. In *Switching Theory in Space Technology.* Stanford University Press, Stanford, CA, 1963.

252. Muller, D. E., and Bartky, W. S. A theory of asynchronous circuits. Proc. of an International Symposium on the Theory of Switching, Cambridge, MA, 1963, pp. 204-243.

253. Myers, J. Analysis of the Simple Code for Data Flow Computation. MIT/LCS/TR-216, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1979.

254. Nadler, D. R. *Data Flow Computer Performance for the GISS Weather Model.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1978.

255. Nadler, D. R. Data Flow Computer Performance for the GISS Weather Model. Computation Structures Group Memo 159, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1978.

256. Oldehoeft, A., Allan, S., Thoreson, S., Retnadhas, C., and Zingg, R. Translation of High Level Programs to Data Flow and their Simulated Execution on a Feedback Interpreter. Technical Report 78-2, Department of Computer Science, Iowa State University, Ames, Iowa, 1978.

257. Patil, S. S. Coordination of Asynchronous Events. TR-72, Project MAC, MIT, Cambridge, MA 02139, June, 1970.

258. Patil, S. S. Synchronizers and Arbiters. Computation Structures Group Memo 165, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1973.

259. Patil, S. S. Bounded and Unbounded Delay Synchronizers and Arbiters. Memo 103, Computation Structures Group, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1974.

260. Patil, S. S. *An Abstract Parallel-processing System.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1967.

261. Patil, S. S. Closure properties of interconnections of determinate systems. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, 1970, pp. 107-116.

262. Patil, S. S. Circuit Implementation of Petri Nets. Computation Structures Group Memo 73, Project MAC, MIT, Cambridge, Mass., December, 1972.

263. Patil, S. S. An Asynchronous Logic Array. MIT/LCS/TM-62, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1975.

264. Patil, S. S. and Dennis, J. B. Description and realization of digital systems. Digest of Papers on Innovative Architecture, IEEE, New York, NY, 1972, pp. 221-233.

265. Peacock, T. Type-checking in Generalized Val. Computation Structures Group Memo, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1983.

266. Peacock, T. Type-checking in Generalized VAL. S.B. thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, May, 1983.

267. Peterson, J. L.. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

268. Petri, C. A. *Kommunication mit Automaten.* Ph.D. Th., Reinisch-Westfalischen Institutes fur Instrumentelle Mathematik, Bonn, 1962.

269. Plummer, W. W. Asynchronous Arbiters. Computation Structures Group Memo 56, Project MAC, MIT, Cambridge, Mass., February, 1971.

270. Presberg, D. L., Saint, H., and Shapiro, R. M. Representation of Algorithms as Cyclic Partial Orderings. Report CA-7112-2711, Vol. I, Applied Data Research, Wakefield, MA, December, 1971.

271. Qualitz, J. E. *Weakly Productive Computation Schemata.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, May 1972.

272. Qualitz, J. E. Decidability of Equivalence for a Class of Data Flow Schemas. MIT/LCS/TM-58, Laboratory for Computer Science, MIT, Cambridge, MA 02139, March, 1975.

273. Ramchandani, C. The computation rate of asynchronous computation systems. Proc. of the Seventh Ann. Princeton Conf. on Information Sciences and Systems, 1973, pp. 276-285.

274. Ramchandani, C. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. MIT/LCS/TR-120, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1974.

275. Ramchandani, C. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1974.

276. Redford, J. L. *A Design for a Routing Module.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1979.

277. Ressler, P. *Simulation of a Highly Parallel Processor.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, January 1979.

278. Ribak, R. *Subsystem Sharing in Parallel Asynchronous Processing.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1968.

279. Ries, P. S. A VLSI Implementation of a Two-by-two Packet Router. Computation Structures Group Memo 197, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1980.

280. Rodriguez, J. E. *A Graph Model for Parallel Computation.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, September 1967.

281. Rodriguez, J. E. A Graph Model for Parallel Computation. MAC/TR-64, Project MAC, MIT, Cambridge, Mass., September, 1969.

282. Rotenberg, L. J. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. MIT/LCS/TR-116, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1974.

283. Rotenberg, L. J. Surveillance mechanisms in a secure computer utility. *ACM Special Interest Group on Computers in Society 2,* 1 (April 1971).

284. Rumbaugh, J. A Parallel Asynchronous Computer Architecture for Data Flow Programs. MIT/LCS/TR-150, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1975.

285. Rumbaugh, J. *A Parallel Asynchronous Computer Architecture for Data Flow Programs.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, May 1975.

286. Singh, N. A Design Methodology for Self-timed Systems. MIT/LCS/TR-258, Laboratory for Computer Science, MIT, Cambridge, MA 02139, February, 1981.

287. Singh, N. *A Design Methodology for Self-timed Systems.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1981.

288. Slutz, D. The Flow Graph Schemata Model of Parallel Computation. MIT/LCS/TR-53, Laboratory for Computer Science, MIT, Cambridge, MA 02139, September, 1968.

289. Slutz, D. *The Flow Graph Schemata Model of Parallel Computation.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, September 1968.

290. Smith, A. *Input/Output in Time-shared, Segmented, Multiprocessor Systems.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, June 1966.

291. Smith, A. Input/Output in Time-shared, Segmented, Multiprocessor Systems. MIT/LCS/TR-28, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1966.

292. Smith, D. Models and Data Structures for Digital Logic Simulation. MIT/LCS/TR-31, Laboratory for Computer Science, MIT, Cambridge, MA 02139, August, 1966.

293. Smith, D. *Models and Data Structures for Digital Logic Simulation.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, August 1966.

294. Sutherland, W. R. *On-line Graphical Specification of Computer Procedures.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, December 1965.

295. Teager, H., Ed. Report of the Long Range Computer Study Group. Massachusetts Institute of Technology, 1961.

296. Tetrick, S. An Instruction Cell Block Design for a Data Flow Computer. Unpublished S.B. thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, 1979.

297. Todd, K. W. Function sharing in a static data flow machine. Proc. of the Intern. Conf. on Parallel Processing, August, 1982, pp. 137-139.

298. Todd, K. W. An Interpreter for Instruction Cells. Computation Structures Group Memo 208, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1981.

299. Todd, K. W. High Level VAL Constructs in a Static Data Flow Machine. MIT/LCS/TR-262, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1981.

300. Todd, K. W. R. *High Level VAL Constructs in a Static Data Flow Machine.* Master's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, February 1981.

301. Tucker, R. *Implementation of Arithmetic for the Data Flow Processing Unit.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, 1980.

302. Tucker, R. Implementation of Arithmetic for the Data Flow Machine Processing Unit. Computation Structures Group Memo 195, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1980.

303. Ullman, J., Aho, A., and Denning, P. J. Principles of optimal page replacement. *Journal of the ACM 18*, 1 (January 1971), 80-93.

304. Vanderbilt, D. Controlled Information Sharing in a Computer Utility. MIT/LCS/TR-67, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1969.

305. van Horn, E. Computer Design for Asynchronously Reproducible Multiprocessing. MAC/TR-34, Project MAC, MIT, Cambridge, MA 02139, November, 1966.

306. van Horn, E. . *Computer Design for Asynchronously Reproducible Multiprocessing.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, November 1966.

307. Vedder, R., Campbell, M., and Tucker, G. The Hughes data flow multiprocessor. Proc. of the Fifth Intern. Conf. on Distributed Computing, New York, NY, 1985, pp. 2-9.

308. Vedder, R, and Finn, D. The Hughes data flow multiprocessor: architecture for efficient signal and data processing. Proc. of the 12th Ann. Intern. Symp. on Computer Architecture, Association for Computing Machinery, New York, NY, 1985, pp. 324-332.

309. Vishniac, E. A Processor Module for a Data Flow Computer. Computation Structures Group Memo 176, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1979.

310. Vishniac, E. M. *A Processor Module for Data Flow Computer Development.* Bachelor's Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, May 1979.

311. Welcome, M. L., and Skedzielewski, S. K. Dataflow graph optimization in IF1. In *Lecture Notes in Computer Science, Volume 201: Functional Programming Languages and Computer Architecture.* Springer-Verlag, Berlin, Heidelberg, New York, 1985, pp. 17-34.

312. Weng, K.-S. Stream-oriented Computation in Recursive Data Flow Schemas. MIT/LCS/TM-68, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October, 1975.

313. Weng, K.-S. An Abstract Implementation for a Generalized Data Flow Language. MIT/LCS/TR-228, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1979.

314. Weng, K.-. *An Abstract Implementation for a Generalized Data Flow Language.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, May 1979.

315. Wetherell, C. S. Error data values in the data flow language VAL. *Transactions on Programming Languages and Systems 4*, 2 (April 1982), 226-238.

316. Yuba, T., T. Shimada, K. Hiraki, and H. Kashiwagi. Sigma-1: A Dataflow Computer For Scientific Computation. Electrotechnical Laboratory, 1-1-4 Umesono, Sakuramura, Niiharigun, Ibaraki 305, Japan, 1984.

317. Zilles, S. N. Bibliography on the Semantics of Programming Languages. Computation Structures Group Memo 75, Project MAC, MIT, Cambridge, MA 02139, March, 1973.