MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

Computation Structures Group Memo 190

Computation Structures Group Progress Report
July 1, 1978 - June 30, 1979

This research was supported in part by the University of California Lawrence Livermore Laboratory under contract 8545403, and in part by the National Science Foundation under grant MCS75-04060 A01

February 1980

COMPUTATION STRUCTURES

Academic Staff

J. B. Dennis, Group Leader
Arvind

Research Staff

W. B. Ackerman

Graduate Students

W. B. Ackerman
D. J. Aoki
S. A. Borkin
G. A. Boughton
J. D. Brock
R. E. Bryant
D. L. Isaman
P. R. Kosinski
J. W. Leth
C. K. C. Leung
L. B. Montz
N. Singh
K.-S. Weng

Undergraduate Students

D. B. Hirschman
P. Hong
J. L. Redford
P. E. Ressler
S. Tetrick
T. L. Tung
E. Vishniac

Support Staff

A. L. Rubin

Visitor

B. Sørbye

A. INTRODUCTION

The Computation Structures Group is playing a central role in the development of computer systems using data driven program execution -- an area that is currently stimulating much interest. Two sessions at the June National Computer Conference [1, 12, 14, 18, 35] in New York were devoted to reports on current data flow research at universities in the United States and abroad. All of these projects and work at several other institutions were represented at the second four-day Data Flow Workshop Conference held at the MIT Endicott House in July 1978. This meeting, hosted by MIT and sponsored by the Department of Energy through the Lawrence Livermore Laboratory, fostered exchange of ideas and discussion of issues by people working in this field of research. A summary of the activities at the workshop has been published [26]. Also a number of professionals from major computer manufacturers attended an MIT summer program, "Data Flow Concepts in Computer Language and Architecture," offered by Professors Arvind and Dennis in June 1979.

In the past year our achievements include: refining the design of our user programming language VAL; deciding on the form and technology of our first data flow computer engineering model; analysis of an example hydrodynamics code to determine its suitability for data flow computation; and completion of a variety of supporting studies in the semantics of data flow language, translation and optimization of data flow programs, fault tolerance of packet communication architecture, routing network structure and performance, generalized implementation of procedures and streams, and the equivalence of formal data models in data base systems.

B. LANGUAGE DESIGN FOR DATA FLOW COMPUTATION

In the past year much effort has been devoted to the design of the programming language VAL. Additionally, a number of theses relating to the semantic specification, optimization, and translation of data flow languages similar to VAL have been completed or are nearing completion. Extensions of "conventional" data flow language have also been pursued. Professor Arvind has investigated a new treatment of streams and arrays whereby much greater asynchrony of program execution is possible. Paul Kosinski has completed a Ph.D thesis in which a semantic model for non-determinate data flow languages is developed.

1. VAL

The Preliminary Reference Manual [2] for the programming language VAL has been published as a LCS technical report. VAL is an applicative language designed for parallel execution of programs. In particular, it is intended for expressing computations such as the numerical solution of partial differential equations, which require efficient parallel execution.

In the design of VAL we have given careful consideration to the recently developed body of knowledge about program structures and language characteristics which support program verification. The natural consistency between language design for

support of concurrency and language design for correctness and verifiability has made it possible to exploit this knowledge in the design of VAL.

We have undertaken the design of a new language because existing languages for numerical computation have a serious deficiency: they reflect the storage structure of the von Neumann concept of computer organization in that each language has some method of effecting a change in state of the memory which cannot be modeled as a local effect. Fortran, still the most popular language for large scale numerical work, is particularly blatant in this respect since it was conceived as a high level notation for programs to be run on a machine of classical design (the IBM 704).

The difficulty with languages that allow specification of global state changes is that programs may be written which are very difficult or impossible to analyze for parts that may be executed concurrently. It is impossible in general to trace the flow of data with less than a complete analysis of the entire program. Only with such analysis is it possible to find and eliminate inessential constraints on the sequencing of program parts.

In contrast, the language VAL is free of side effects: each module or well formed portion of a VAL program corresponds to a mathematical function and the entire effect of putting two parts together is to compose the corresponding functions. Such a language is *functional* or *applicative*. Although designs for applicative languages have been discussed many times in the literature, there have been few attempts to construct a complete and practical definition. This is due to the difficulty of incorporating file updates and input/output operations within the applicative framework, and the problems of efficient implementation. The file update and input/output issues will be addressed in future versions of VAL in which streams of values [5, 36] will be introduced as a principal means for communicating between program modules. The efficiency issue is countered by our goal of developing computer architectures for efficient, highly parallel execution of programs expressed in functional languages.

In developing the structure of VAL, it was natural for us to start from a language design which is of high quality, is well documented, and is close in spirit to our goals. Such a language is CLU [23, 24], developed at LCS by the Programming Methodology Group under Professor Barbara Liskov. In particular, CLU is designed for complete compile time type checking and has a set of well thought-out control structures and basic data types consonant with modern principles of structured programming.

While we have adopted many fundamental ideas of CLU, VAL differs radically from CLU in that the latter, like most programming languages, is object-oriented instead of value-oriented. In keeping with this difference, the syntax and general structure of VAL are designed to reflect the functional character of the language and our desire to support highly concurrent program execution.

One of the the principal features of VAL is the forall program structure, to perform operations independently on many elements of an array. For example, if A and B are vectors of length 1000, their element-by-element sum may be computed by

```
C := forall J in [1, 1000]
       construct A[J] + B[J]
     endall
```

and their inner product by

```
N :=forall J in [1, 1000]
       eval plus A[J] * B[J]
     endall
```

Several example programs have been expressed in VAL: a numerical algorithm provided by the Lawrence Livermore Laboratory, and several class projects written by students in an MIT graduate subject in data flow computer architecture during the Fall term 1978.

In the Summer of 1979 a programming system for VAL consisting of a translator and an interpreter will be developed using the CLU programming system. The immediate objective is to permit development and testing of VAL programs. However, the translator will be used later in generating machine code for simulation studies of proposed data flow systems.

2. Correct Translation of a Data Flow Language

In his master's thesis, J. D. Brock [8] gives a two-step operational semantics for ADFL, an Applicative Data Flow Language similar to VAL. The first step is performed by an algorithm for translating data flow programs into data flow graphs. For the second step, each data flow operator is characterized as a function mapping input histories into output histories. With these functions, Kahn's fixpoint theory for communicating processes [17] may be used to derive the result of graph execution.

In another paper [9], Brock presents a denotational semantics for ADFL and proves it consistent with the operational semantics. The denotational and operational semantics are not equivalent. The denotational semantics specify that expression evaluation must terminate to yield results and that, if expression evaluation terminates, all sub-expression evaluations terminate. However, in data flow, and many other models of concurrent computation, a computation may produce results even if some sub-computations do not terminate. The characterization of such computations contributes much to the complexity of the operational semantics of ADFL. Consequently, the simpler denotational semantics are the more useful in tasks such as program verification. The proof of consistency assures those using the simpler semantics that the two semantic theories agree on all "denotationally" terminating expression evaluations.

3. Data Flow Program Optimization

The correctness of Brock's translation algorithm depends on viewing the arcs of program graphs as unbounded FIFO queues between data flow operators. In her master's thesis, Lynn Montz [27] investigates the replacement of these unbounded queues with buffers of length one. This decision is a reflection of the design of the instruction cells

and routing network of the Dennis-Misunas data flow machine [13] for which this translation is aimed. The implication of the length one arcs is that actors must be prevented from producing new tokens until their output arcs are empty. This behavior is ensured by redefining the firing rules so that no operator is enabled if a token is present on any of its output arcs. While the resulting behavior might occasionally be displayed by "infinite queue" graphs, the changed firing rule produces an equivalent set of graphs constrained so that this desired behavior is, in fact, the only possible behavior.

By performing a transformation which replaces each arc of the graph by an appropriate data/acknowledge arc pair (d/a arc pair), the token flow constraints defined by the firing rules can be explicitly built into the graph: The presence of a token on the acknowledge arc of a d/a arc pair signals the emptying of the data arc. Though intuitively the token flow constraint imposed by the transformation seems proper, it is necessary to show that it introduces no new source of deadlock. Such a situation might arise in the event that some cycle within a graph becomes "full" of tokens. In the thesis the equivalence of "infinite queue" graphs and "data/acknowledge arc" graphs is established.

While the acknowledge arc permits the safe functioning of data flow graphs, it does not provide a "free" solution. Aside from the obvious overhead involved in incorporating acknowledge arcs and tokens, the constraints which they impose on the token flow through the graphs may cause bottlenecks. In response to these issues, optimization techniques have been developed which are specifically aimed at either decreasing the overhead by removal of unnecessary acknowledge arcs or increasing the throughput by balancing token flow.

4. Computer Representation of Data Flow Graphs

Jim Leth's master's thesis, entitled An Intermediate Form for Data Flow Programs [20], is expected to be complete in mid-July. The thesis proposes a representation for data flow graphs as an abstract data type (cluster) implemented in the language CLU. Cluster operations are provided for building graphs by connecting the input and output ports of operators. The resulting graphs can themselves be treated as operators and used to form more complex graphs. A mechanism for assigning names to operator inputs and outputs is used to support the identifier binding process in the data flow source language VAL. Using this mechanism, a scheme is presented for translating VAL programs into the CLU graph representation using the cluster operations.

Finally, the thesis briefly discusses the other phases of compilation of VAL source programs into data flow machine code, with emphasis on how the proposed intermediate form can satisfy the requirements of optimization and code generation.

5. Data Flow Machine Language

The master's thesis of Don Aoki [3] is a specification of a machine language instruction set for a simple data flow computer. This work involves the determination of the technique used in providing acknowledge signal generation. Whenever an instruction

cell fires, it normally sends an acknowledge signal to each instruction cell which has sent it an operand. There are two distinct approaches to acknowledge signal generation: implicit generation and explicit generation.

The implicit generation approach requires the memory address of an instruction cell be included as a tag in each operation packet it transmits. This tag is forwarded in each result packet produced by instruction execution. The target instruction uses the tags for its operands to automatically generate acknowledge packets. The explicit generation method includes special destinations in each instruction that specify instructions to which acknowledge signals are sent.

It appears that explicit generation should be used because it permits, as an optimization, minimization of the number of acknowledge signals required for ensuring deadlock-free behavior. Implicit generation has the advantage of permitting multiple operand sources, but this advantage is offset by the potential decrease in routing network performance due to unnecessary acknowledge packets, and the increase in operation packet length.

Other issues to consider will be determining what scalar types will be allowed, how they will be represented, and the choice of instruction cell parameters such as the number of operand receivers and destination fields.

6. Generalized Streams and I-Structures

Professor Arvind has proposed that data flow streams [5, 36] be implemented as "structures with holes," structures which have missing values at some selectors. Each missing value is actually a pointer to a site where a data flow computation is producing the value. Implementation of structures with holes is straightforward if structures are implemented using global memory.

Earlier, our intended implementation of streams was a history of values passed through the links of a data flow graph. The stream operation cons(a,'x) was performed by producing the value a followed by the elements of the stream x. In the structure with holes implementation, a generalized stream is represented by a single data flow token pointing into a global memory. The stream operation cons(a,'x) is performed as soon as either a or x is available. The produced stream token will contain a hole for the missing value.

There is a natural away to extend the implementation of generalized streams to I-structures, a "generalized" array implemented as a structure with holes. I-structures offer as much asynchrony as generalized stream, and at the same time preserve the ease of coding implicit in array manipulations.

7. Semantics of Non-determinite Languages

Paul Kosinski has completed his Ph.D thesis entitled Denotational Semantics of Determinate and Non-determinate Data Flow Programs [19]. This research, detailed in a previous progress report [10], defines a denotational semantics for a data flow language which has streams as a basic data type and an Arbiter, a data flow operator that non-determinately merges two streams into one.

C. DESIGN FOR DATA FLOW COMPUTATION

We have completed the design of a prototype data flow computer system to be constructed by September 1980. In addition, several supporting studies relating to the hardware realization of data flow computation are being investigated by students as thesis projects. These areas include design methodology and fault tolerance techniques for self-timed systems, machine structures supporting high-level data flow languages, and the analysis, with complexity, simulation, and logical models, of machine structures.

1. Engineering Model

In data flow architecture, our primary effort is developing prototype data flow computer systems so their cost and performance may be evaluated and the problems of effectively using them may be assessed. Two forms of data flow computer which use fixed allocation of instructions to physical memory cells are of immediate interest for large-scale scientific problems: the Cell Block machine illustrated in Fig. 1 which has evolved from the ideas of Dennis and Misunas [13], and the Data Flow Multiprocessor shown in Fig. 2 which is related to Rumbaugh's design [32, 33] and is similar to an experimental data flow machine developed by the Texas Instruments Corporation [16].

Fig. 1. Cell Block Data Flow Machine

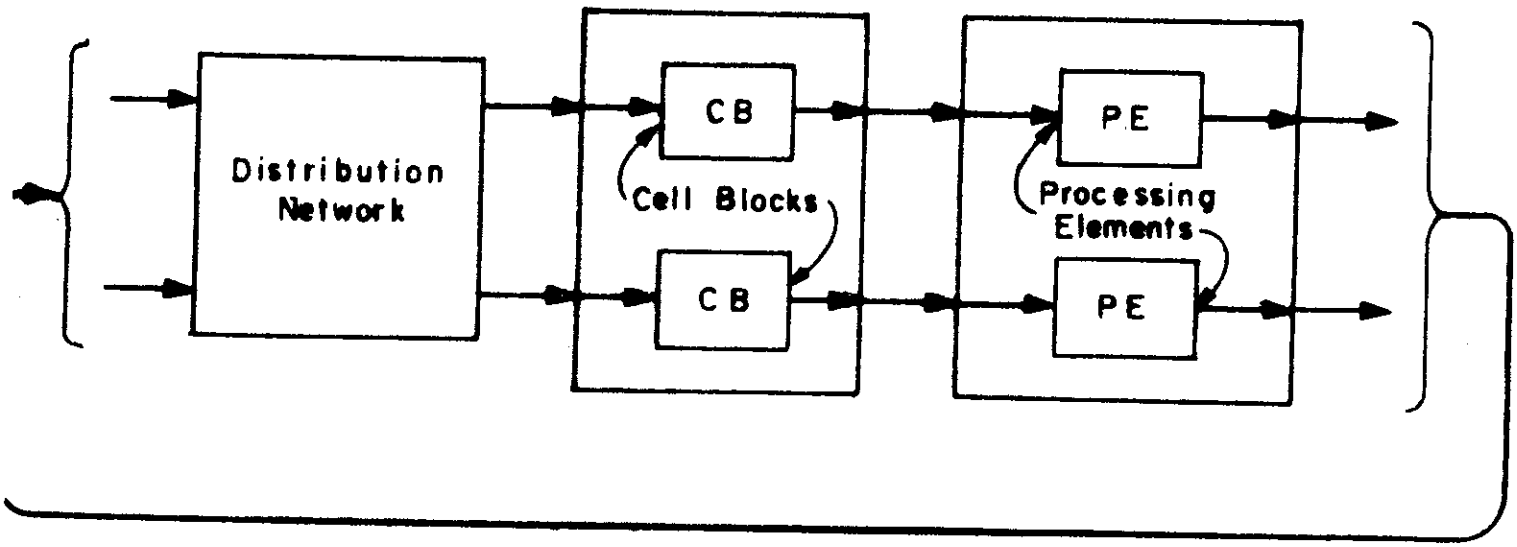
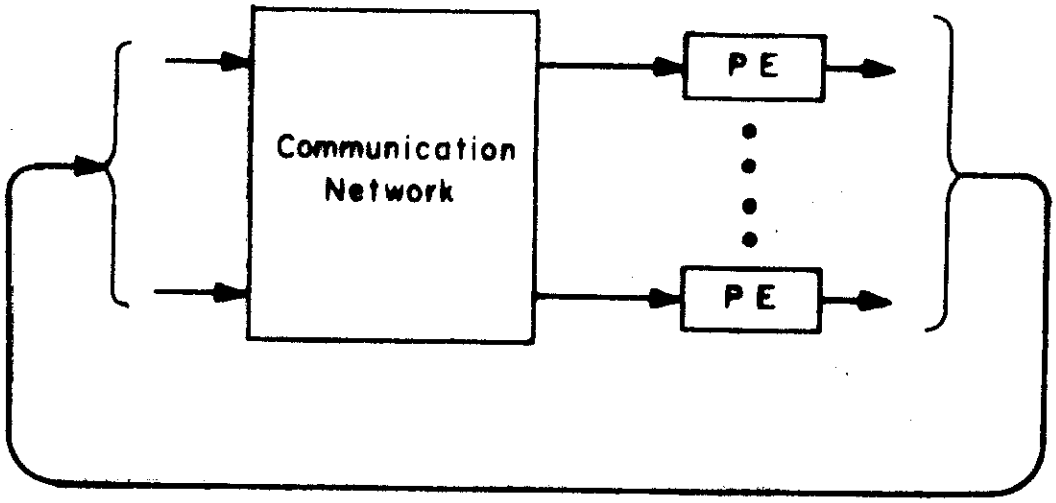


Fig. 2. Data Flow Multiprocessor



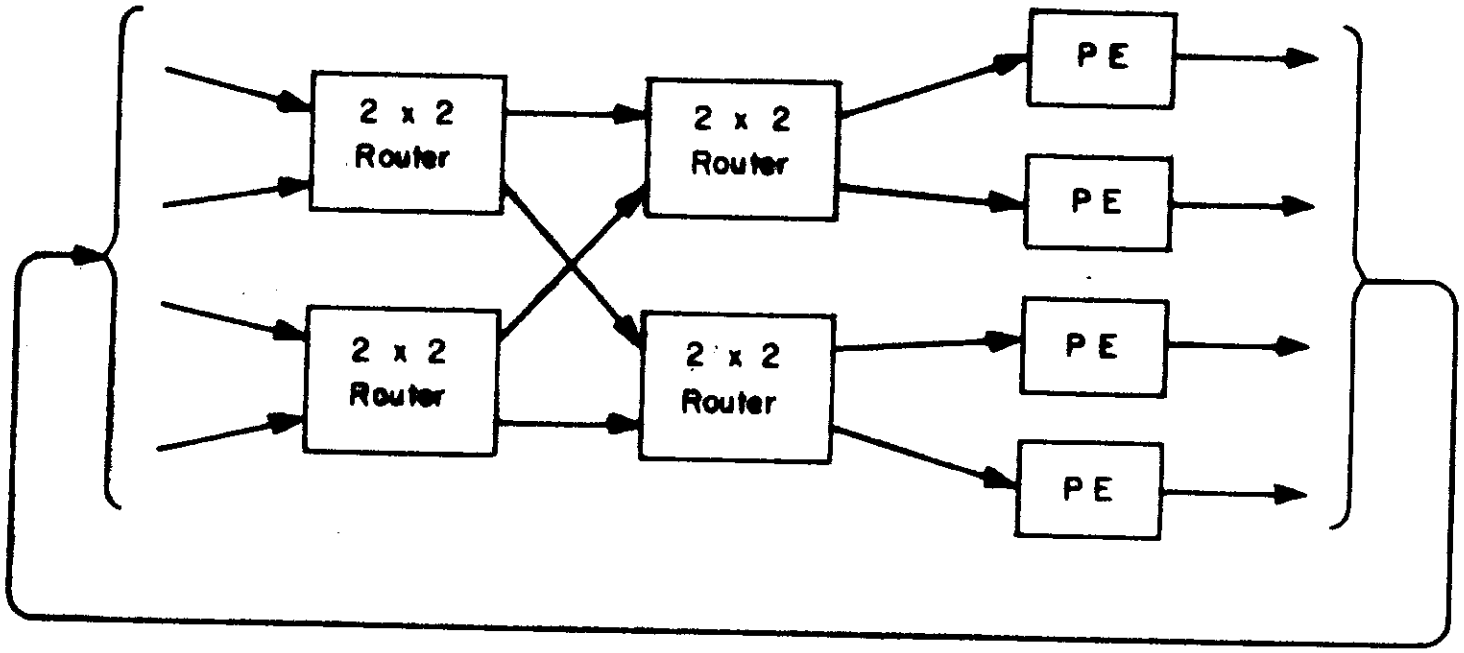
Our approach to building data flow prototypes is to design two basic module types: a processor module and a communication module.

The processor module will be microprogrammable so it can emulate various units of data flow systems such as a Cell Block, a functional unit or a complete processing element. The communication module will be a 2 x 2 router from which routing networks as large as needed can be built. A variety of packet communication architectures, including the two mentioned above, may be realized by appropriate assembly of these modules. Our first engineering model of a data flow machine will be a Cell Block machine realized using four processor and four routing modules as in Fig. 3.

2. Processor Module

Ephraim M. Vishniac has completed his B. S. thesis entitled A Processor Module for Data Flow Computer Development [34]. This thesis presents a design for a general purpose computer module for use in prototype Data Flow machines. The computer module is an eight-bit-wide programmable microprocessor designed from bit-slice LSI components, and has input and output packet ports which allow byte-serial packet transmission between modules. The computer module can be microprogrammed to emulate various modules of a Data Flow Machine. For example, in our first engineering model, Fig. 3, a single computer module will implement a Cell Block, Processor Unit pair.

Fig. 3. MIT Engineering Model



We are presently completing a detailed logic design for a computer module similar to the Vishniac module. The new design incorporates additional hardware features which facilitate the diagnosis of hardware faults and the initial loading of microcode and data. The design will be realized on a wire wrap prototype board and the SUDS (Standford University Drawing System) design automation system is being used to generate the wire list. Support software for the module including an assembler and a simulator are also being developed.

3. Router Module

The 2 X 2 router is a simple building block which can be used to construct classes of routing networks with different topologies and operational characteristics.

A 2 X 2 router receives packets at its two input ports and delivers each received packet at one of two output ports according to a destination address carried by the packet. Each packet is transmitted byte-serially between modules. Packet bytes are delivered and received using an asynchronous packet communication protocol.

Each 2 X 2 router module is designed so that packets to be forwarded at different output ports can be processed concurrently (Fig. 4). Such concurrency is naturally supported by decomposing the router into two input modules (IM) and two output modules (OM) (Fig. 5). Each input module is a sequential machine which examines the destination address in each input packet, makes an output request to the specified output module and delivers the packet bytes when the request is granted. Each output

Fig. 4. Parallel Processing in a 2 x 2 Router

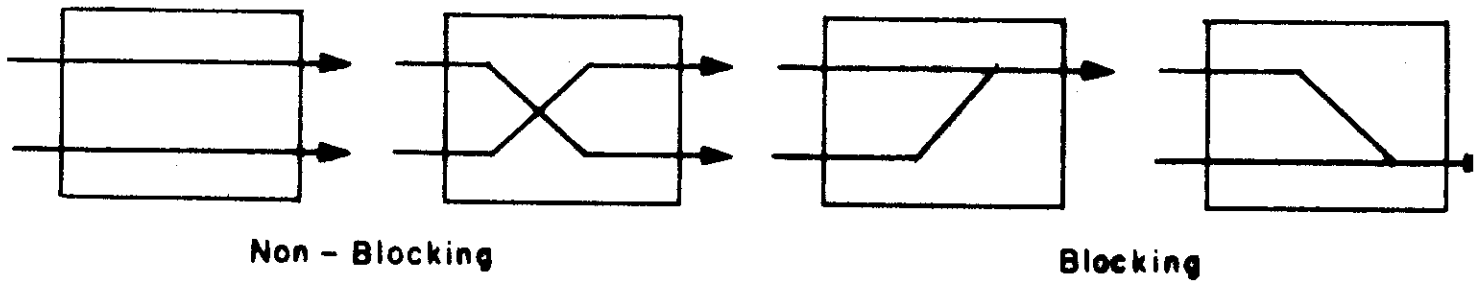
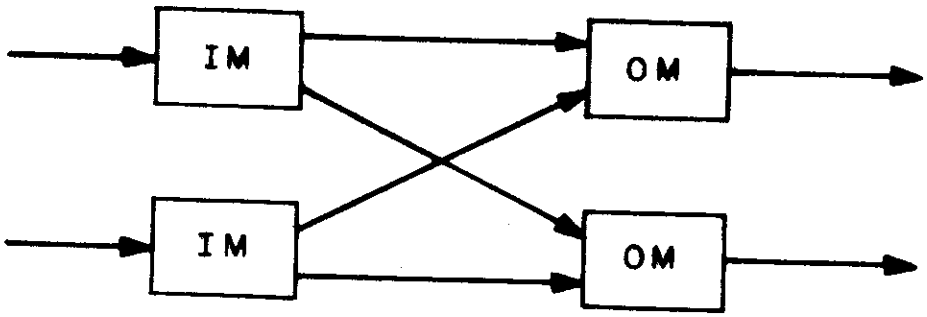


Fig. 5. Structure of a Router Module



module consists of a data multiplexer and an arbiter to resolve conflicts between output requests. A logic design by T. L. Tung of the 2 X 2 router module based on this modular structure has been completed. The asynchronous sequential circuits and arbiters used in this design are constructed using SSI components and discrete transistors. The data paths consist of MSI multiplexers and SSI logic gates.

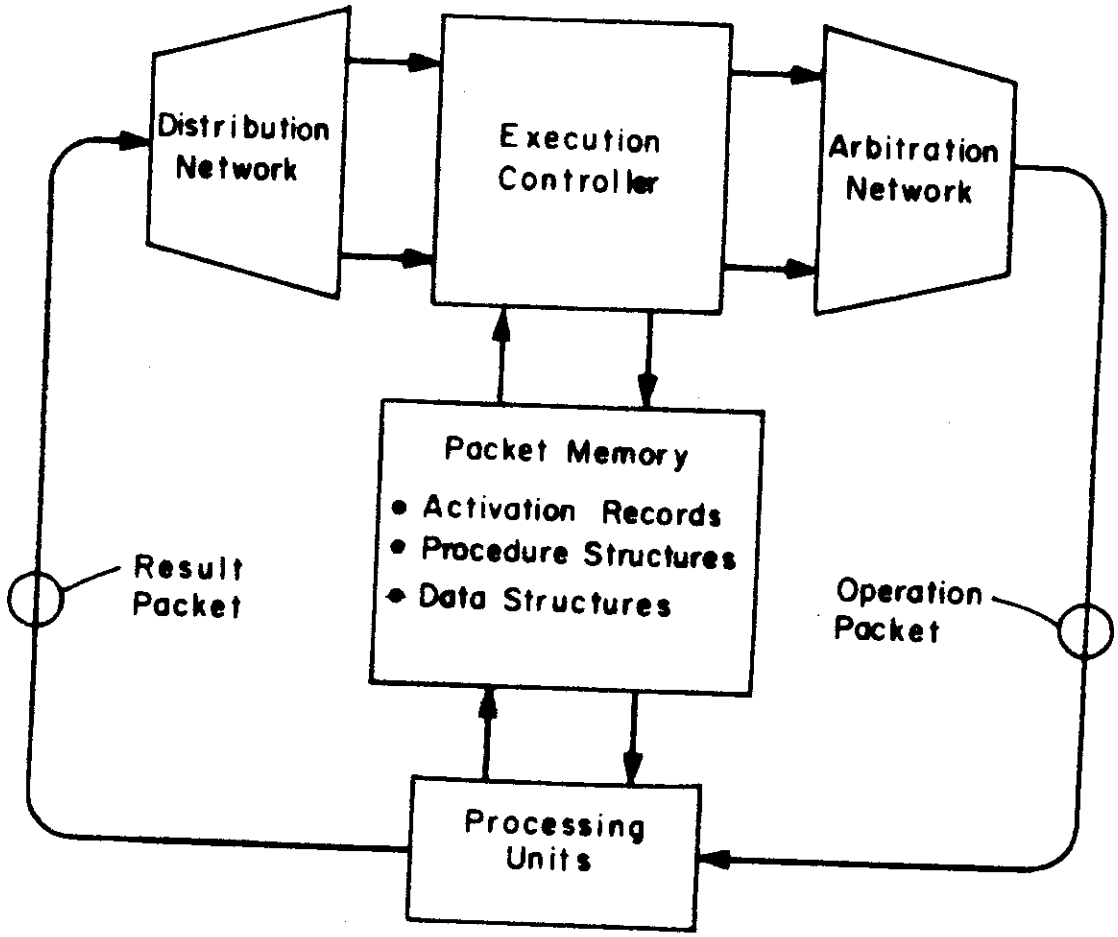
4. Abstract Data Flow Machines

For a feasibility study of a general purpose data flow processor supporting dynamic instruction allocation, Weng has chosen a value-oriented language that incorporates stream values and a forall construct. The inclusion of stream values is an attractive approach to increase the expressiveness of VAL so input/output processing may be programmed without introducing side effects or nondeterminacy. In his doctoral thesis, Weng [37] has chosen to implement this language on an architecture that supports recursive data flow schemas. This leads to a simple mechanism for procedure activation

and removes the need for the acknowledge signals required in a data flow machine using static instruction allocation.

This processor (Fig. 6) differs from the Cell Block machine in the design of the Execution Controller and the Packet Memory. The Execution Controller functions much like the Activation Memory of the simpler machine except that it accesses the Packet Memory for storage and retrieval of activation records and instructions. The Packet Memory holds data structures, procedure structures, and activation records. The implementation is designed so activation records grow only to the size required for those instructions actually executed in a given procedure activation.

Fig. 6. Data Flow Processor



Stream values are implemented as data structures, similar to Arvind's I-structures, whose components may be "holes." This solution is easily extendable to work for streams of streams and removes the inefficiency discussed by Weng [36]. We propose two forms of forall. One form allows one to express concurrent computations on an index range of an array, and the second form allows exploitation of concurrency when the array may be very sparsely defined.

5. Self-timed Hardware Systems Design

Leung has initiated an effort to develop methodologies for designing and implementing self-timed hardware systems organized by a packet communication architecture. Packet communication systems are designed and described using ADL, an architecture description language [21] we have used in describing several forms of data flow processors. A self-timed hardware module receives and delivers packets and acknowledgments via asynchronous packet communication protocols. Techniques for implementing ADL descriptions with self-timed modules fall naturally into two classes: for implementing combinational and sequential functions with self-timed modules using specific packet protocols, and for connecting a number of modules together to perform other tasks.

In our work we have developed a two input AND-gate for dual-rail protocols and have used it to implement combinational functions in self-timed modules. The set of basic modules includes a pipelined register module, which is also used in feedback paths to implement sequential functions, a switch module, a multiplexor module and a non-deterministic merge module. It is straightforward to translate ADL constructs into data flow schemas and then implement nodes in a schema with modules from this set, assuming the availability of operator modules such as adders and equality testers. The approach is demonstrated with a 2 X 2 router design and is being evaluated through comparison with Tung's router design, which uses commercially available multiplexor chips controlled by a number of asynchronous sequential circuits and an arbiter circuit, and with Redford's [30] design, which is derived from a Petri net specification of the router.

6. Fault Tolerance

In his thesis research, Leung [22] uses a random pulse train model for hardware failures in self-timed modules to study fault tolerance techniques. A hardware module outputting random pulse trains may violate the adopted packet communication protocols, deliver erroneous packets, or hang up communication channels. Pathological interactions between random pulse trains and dual-rail protocols has been studied in detail. A hardware module is operating improperly if it violates the adopted packet protocols in delivering packets and acknowledgments. A filtering register has been designed for dual-rail protocols to sustain proper operation in spite of random pulse train inputs. This register is useful for eliminating several types of pathological input conditions to facilitate the design of fault detectors and voters.

A self-checking module is a hardware module whose inputs and outputs are coded for error detection. Schemes to represent conventional error-detecting codes in dual-rail protocols have also been developed and used to construct self-checking self-timed modules. These self-checking modules are implemented with submodules such that no single submodule failure can defeat the error-detecting capability of the chosen code. Erroneous packets due to single submodule failures can then be detected by checking module outputs for code violations. If a self-checking module is embedded in a packet communication architecture which delivers identical inputs, to each of its submodules within a bounded time interval, and if the submodules take approximately the same amount of time to process identical inputs, then hardware failures which hang up communication channels can also be detected.

A packet communication system designed to facilitate fault detection should not miss failure symptoms nor generate false alarms. It should provide diagnostics information and should maintain data integrity in the presence of hardware failures. System strategies to achieve these goals are currently under investigation.

7. Analysis of Routing Networks

During the past year we have continued our analysis of routing networks. A routing network accepts tagged packets on its input ports and directs each packet to the output port associated with the packet's tag. A primary area of study has been a class of routing networks based on the indirect binary n-cube topology [29]. This topology is interesting because it leads to N-input networks with only $N \log_2 N$ internal nodes, and it permits use of a very simple routing algorithm at each node. This study has produced an abstract model of these networks which is considerably easier to analyze than the networks themselves, and for an even distribution of packet tags predicts a level of network performance that agrees well with simulation results. This model is presently being used to predict the performance of very large networks of this class, and to examine the effect of buffer size on network performance.

Another area of study has been the cost of wires required to implement various routing network structures. While more work remains to be done, it is clear that there is a fundamental relation between the complexity of a communication problem and the amount of wire needed to solve it.

8. Simulation of a Simple Data Flow Machine

Paul Ressler [31] has recently completed a bachelor's thesis on the simulation of a simple data flow machine. In the thesis, a simulator mechanism is described which is suitable for modeling packet flow in asynchronous packet switching architectures. An implementation of the simulator mechanism is discussed and is used to specify a data flow machine architecture. An analysis of the behavior of a Fast Fourier Transform program using the simulator is presented.

9. Logical Models for MOS LSI

As large scale integrated circuits become more economical and more sophisticated in their capabilities, custom LSI will become an attractive means for implementing specialized digital systems. Currently, LSI circuits are hand designed and carefully optimized by highly trained technicians. In the future, however, nonspecialists will design their own circuits with more modest densities and speeds. This trend has already begun at several universities and research laboratories around the country [25]. Custom LSI is particularly attractive for our data flow machine work, because our hardware needs are not well served by current commercial products. Computerized tools can greatly simplify the design process and thereby bring the field of LSI design into the realm of the nonspecialist.

R. Bryant has begun research in developing logic design tools specifically for MOS integrated circuits. As a first step, he has been investigating logical models which more closely match the circuit elements of MOS LSI than do more traditional logic gate models. Based on this model, a program has been written which provides a three-level logic simulation for nMOS circuits. Because it directly models the actual circuit elements, the program can accurately and consistently represent such features as dynamic storage, pass transistors, and errors caused by the bilateral properties of the transistors. Furthermore, the description of the circuit used by the simulator could easily be derived directly from the IC mask specification.

D. DATA FLOW APPLICATIONS

Partial differential equation computation has often been proposed as an ideal area for the application of highly concurrent computer architectures. The high computational requirements of these problems provide an incentive for high speed computation, while the regularity and minimal data dependencies provide hope that this speed can be achieved through parallelism.

To examine some of the issues in exploiting the concurrency of partial differential equation simulations, the Computation Structures Group has made two studies of the SIMPLE code [11], a 1500 line FORTRAN program developed at Lawrence Livermore Laboratories. This program provides a typical, although condensed example of a simulation of hydrodynamic motion and heat flow in a compressible fluid.

1. Data Flow Computer Structures for Partial Differential Equation Simulation

Arvind and R. Bryant [4] have found that while SIMPLE exhibits a great deal of regularity, i.e., homogenity of the computation over the entire grid and for the most part only "nearest-neighbor" data dependencies, the regularity is not perfect and the data dependencies vary dynamically. Hence, any highly concurrent computer system which relies on absolute regularity or a static allocation of computing resources will achieve only limited success in terms of both programmability and performance.

The numerical methods used by SIMPLE to approximate the partial differential equations were of course developed in the context of sequential computers. New types of numerical methods may be developed which provide greater regularity and more potential concurrency, but these developments will come too slowly to "rescue" any architecture which cannot achieve reasonable programmability and performance with existing methods. Furthermore, such factors as boundary region calculations, and data dependent decisions will always cause irregularities in the program structure. Partial differential equation simulations seem to call for greater flexibility in both programming and execution than has generally been acknowledged.

As a first step in providing this flexibility, data flow languages [1, 2, 5] provide an attractive means for expressing PDE simulation programs. These languages display the potential concurrency of a program without constraining its implementation to a particular system configuration or timing model.

A computer which executes data flow programs must then map the program onto the processing resources in such a way that many activities proceed concurrently. We are currently investigating several design decisions for a data flow computer architecture and their impact on programmability, performance, and cost. There seem to be several alternative mechanisms for resource allocation and communication which provide tradeoffs between the three goals.

2. Role Diagram Analysis of the SIMPLE Code

John Myers [28] has analyzed SIMPLE from the standpoint of computation on a data flow computer that is not yet fully specified, with the objectives of helping to further specify the computer and to develop VAL as its source language.

The algorithm viewed as "abstract" (i.e., independent of physical arrangements in space and time for its realization) is shown to imply spatial and temporal structure that will have to appear in any and all implementations. Expressing this structure in a form useful for either hardware design or program compilation requires grosser levels of description, with the grosser levels reflecting modularity of computing resources conjoined with modularity of the algorithm. Following Holt [15] we use role diagrams to display spatio-temporal structure that reflects both the problem and possible resource arrangements. These diagrams guide translation into VAL and the analysis of required computation time.

Except for the output of results, the use of N processors configured as a data flow computer can reduce the time to compute the SIMPLE code by a factor of at least $N^{1/2}$. Sequencing constraints that limit improvement to this factor occur in the calculation of heat flow. These constraints stem from the method chosen in the SIMPLE code for inversion of a tri-diagonal matrix: back-substitution. It would appear feasible to find or develop a method with weaker sequencing constraints so that all phases of the problem could be computed in times that would decrease by a factor of N/log N as N increases.

In view of the large size of hydrodynamics problems, efficient use of computing resources will be required. Efficient compilation must use higher-level descriptions of the problem, in addition to a data flow graph at the level of machine instructions. We discuss research aimed at bringing under control the expression of the space-time aspect of an algorithm at different levels of detail, so as to guide transformation of the algorithm toward an efficient machine program for a particular computer organization.

E. DATA BASE MODELS

In his Ph.D thesis [7], S. Borkin examines several equivalence properties for data models. A data model defines the types of structures present in a database and the types of operations which may be used to alter the database. An understanding of data model equivalence properties is necessary if one wishes to implement a system which presents different users with views of a database in terms of differing data models or which provides a common interface to several database systems defined in terms of different data models. Using formal definitions of the terms database, operation, operation type, application model and data model, equivalence according to state, operation, application model, and data model are distinguished. Three types of application and data model equivalence may be defined - isomorphic, composed operation and state dependent.

Semantic data models are data models whose structures are meant to have clear interpretations in terms of the applications being modeled. In the thesis, the semantic graph and semantic relation data models are formally defined. The definition of the semantic relation data model includes the definitions of constraints and the semantic relational algebra. It is proved that the semantic graph and the restricted semantic relation data models are state dependent equivalent. Observations on the network vs. relational data model "controversy" are presented.

Much of the content of this thesis has been published in a paper presented at the International Conference on Very Large Data Bases [6].

REFERENCES

1. Ackerman, William B. "Data Flow Languages." Proceedings of the 1979 National Computer Conference, AFIPS Conference Proceedings, Vol. 48 (June 1979), 1087-1095. Also M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 177. Cambridge, Ma., May 1979.

2. Ackerman, William B., and Dennis, Jack B. VAL -- A Value-Oriented Algorithmic Language: Preliminary Reference Manual. M.I.T., Laboratory for Computer Science, LCS/TR-218. Cambridge, Ma., June 1979.

3. Aoki, Donald J. A Machine Language Instruction Set for a Data Flow Processor. S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, forthcoming.

4. Arvind, and Bryant, Randal E. "Parallel Computers for Partial Differential Equation Simulation." To appear in Proceedings of Scientific Computer Information Exchange Meeting, September 1979.

5. Arvind, Gostelow, Kim B., and Plouffe, William. The (Preliminary) Id Report. University of California, Department of Information and Computer Science, TR 114. Irvine, Ca., May 1978.

6. Borkin, Sheldon A. "Data Model Equivalence." Fourth International Conference on Very Large Data Bases, September 1978, 526-534.

7. Borkin, Sheldon A. Equivalence Properties of Semantic Data Models for Database Systems. M.I.T., Laboratory for Computer Science, LCS/TR-206. Cambridge, Ma., January 1979.

8. Brock, J. Dean. Operational Semantics of a Data Flow Language. M.I.T., Laboratory for Computer Science, LCS/TM-120. Cambridge, Ma., December 1978.

9. Brock, J. Dean. Consistent Semantics for a Data Flow Language. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 172. Cambridge, Ma., January 1979.

10. Computation Structures Group. Progress Report July 1976 - July 1977. M.I.T., Laboratory for Computer Science, LCS/PR-XIV. Cambridge, Ma., 31-32.

11. Crowley, W. P.; Hendrickson, Christopher P.; and Rudy, Timothy E. The SIMPLE Code. Lawrence Livermore Laboratories, Internal Report UCID-17715. Livermore, Ca., February, 1978.

12. Davis, Alan L. "A Data Flow Evaluation System Based on the Concept of Recursive Locality." Proceedings of the 1979 National Computer Conference, AFIPS Conference Proceedings, Vol. 48 (June 1979), 1079-1086.

13. Dennis, Jack B., and Misunas, David P. "A Preliminary Architecture for a Basic Data-Flow Processor." Conference Proceedings of the Second Annual Symposium on Computer Architecture, January 1975, 126-132.

14. Gostelow, Kim P., and Thomas, Robert E. "A View of Dataflow." Proceedings of the 1979 National Computer Conference, AFIPS Conference Proceedings, Vol. 48 (June 1979), 629-636.

15. Holt, Anatol W. Roles and Activities, a System of Describing Systems. Academic Computing Center, Boston University. Boston, Ma., 1979.

16. Johnson, D., et al. "Automatic Partitioning of Programs in Multiprocessor Systems." To appear in Proceedings of Spring Comcom 80, February 1980.

17. Kahn, Gilles. "The Semantics of a Simple Language for Parallel Programming." Information Processing 74. New York: American Elsevier, 1974.

18. Keller, Robert M.; Lindstrom, Gary; and Patil, Suhas S. "A Loosely-Coupled Applicative Multi-processing System." Proceedings of the 1979 National Computer Conference, AFIPS Conference Proceedings, Vol. 48 (June 1979), 613-622.

19. Kosinski, Paul R. Denotational Semantics of Determinate and Non-determinate Data Flow Programs. M.I.T., Laboratory for Computer Science, LCS/TR-220. Cambridge, Ma., July 1979.

20. Leth, James W. An Intermediate for Data Flow Programs. S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, forthcoming.

21. Leung, Clement C. K. C. "ADL: An Architecture Description Language for Packet Communication Systems." To appear in Proceedings of the 4th International Symposium on Computer Hardware Description Languages.

22. Leung, C. K. C. Fault Tolerance in Packet Communication Computer Architecture. Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, forthcoming.

23. Liskov, Barbara H.; Snyder, L. Alan; Atkinson, Russell R.; and Schaffert, J. Craig. "Abstraction Mechanisms in CLU." Communications of the ACM Vol. 20 No. 8 (August 1977), 564-576.

24. Liskov, Barbara H.; Moss, J. Eliot; Schaffert, J. Craig; Scheifler, Robert W.; and Snyder, L. Alan. CLU Reference Manual. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 161. Cambridge, Ma., July 1978.

25. Mead, Carver, and Conway, Lynn. Introduction to VLSI Systems. Addison-Wesley, 1979.

26. Misunas, David P. Report on the Second Workshop on Data Flow Computer and Program Organization. M.I.T., Laboratory for Computer Science, LCS/TM-136. Cambridge, Ma., May 1979.

27. Montz, Lynn B. Safety and Optimization Transformations for Data Flow Programs. S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, forthcoming.

28. Myers, John. Analysis of the SIMPLE Code for Data Flow Computation. M.I.T., Laboratory for Computer Science, LCS/TR-216. Cambridge, Ma., May 1979.

29. Pease, Marshall C. "The Indirect Binary N-Cube Microprocessor Array." IEEE Transactions on Computers, Vol. C-26 No. 5 (May 1977), 458-473.

30. Redford, John L. "A Design for a Routing Module." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1979.

31. Ressler, Paul E. "Simulation of a Highly Parallel Processor." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1979.

32. Rumbaugh, James E. A Parallel Asynchronous Computer Architecture for Data Flow Programs. M.I.T., Laboratory for Computer Science, LCS/TR-150. Cambridge, Ma., May 1975.

33. Rumbaugh, James E. "A Data Flow Multiprocessor." IEEE Transactions in Computers, Vol. C-26 No. 2 (February 1977), 138-146.

34. Vishniac, Ephraim M. A Processor Module for Data Flow Computer Development. M.I.T., Laboratory for Computer Science, Computation Structure Group, Memo 176. Cambridge, Ma., May 1979.

35. Watson, Ian, and Gurd, John. "A Prototype Data Flow Computer with Token Labelling." Proceedings of the 1979 National Computer Conference, AFIPS Conference Proceedings, Vol. 48 (June 1979), 623-628.

36. Weng, Kung-Song. Stream-Oriented Computation in Recursive Data Flow Schemes. M.I.T., Laboratory for Computer Science, LCS/TM-68. Cambridge, Ma., October 1975.

37. Weng, Kung-Song. An Abstract Implementation for a Generalized Data Flow Language. Ph. D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, forthcoming.

Publications

1. Ackerman, William B. "A Structure Processing Facility for Data Flow Computers." Proceedings of the 1978 International Conference on Parallel Processing, August 1978. Also M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 165. Cambridge, Ma., July 1978.

2. Ackerman, William B. "Data Flow Languages." Proceedings of the ACM 1979 National Computer Conference. Also M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 177. Cambridge, Ma. May 1979.

3. Ackerman, William B., and Dennis, Jack B. VAL -- A Value-Oriented Algorithmic Language, Preliminary Reference Manual. M.I.T., Laboratory for Computer Science, LCS/TR-218. Cambridge, Ma., June 1979.

4. Borkin, Sheldon A. "Data Model Equivalence." Proceedings of the Fourth International Conference on Very Large Data Bases. New York: ACM. Also M.I.T., Laboratory for Computer Science, LCS/TM-118. Cambridge, Ma., December 1978.

5. Borkin, Sheldon A. Equivalence Properties of Semantic Data Models for Database Systems. M.I.T., Laboratory for Computer Science, LCS/TR-206. Cambridge, Ma., January 1979.

6. Brock, J. Dean. Operational Semantics of a Data Flow Language. M.I.T., Laboratory for Computer Science, LCS/TM-120. Cambridge, Ma., December 1978.

7. Brock, J. Dean. Consistent Semantics for a Data Flow Language. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 172. Cambridge, Ma., January 1979.

8. Bryant, Randal E., and Dennis, Jack B. "Concurrent Programming." Research Directions in Software Technology. Edited by P. Wegner. Cambridge, Ma.: The MIT Press, June 1979, 584-610.

9. Denning, Peter J.; Dennis, Jack B.; and Qualitz, Joseph E. Machines, Languages, and Computation. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1979.

10. Dennis, Jack B. Associate Editor, Research Directions in Software Technology. Edited by P. Wegner. Cambridge, Ma.: The MIT Press, June 1979.

11. Dennis, Jack B. "Introduction to Part II. 2 Computer Systems Methodology." Research Directions in Software Technology. Edited by P, Wegner. Cambridge, Ma.: The MIT Press, June 1979, 421-424.

12. Dennis, Jack B.; Fuller, Samuel H.; Ackerman, William B.; Swan, Richard J.;and Weng, Kung-Song. "Research Directions in Computer Architecture." Research Directions in Software Technology. Edited by P. Wegner. Cambridge, Ma.: The MIT Press, June 1979, 514-556.

13. Feridun, Arif. Design of an On-Line Byte-Level Pipelined Arithmetic Processor. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 162. Cambridge, Ma., July 1978.

14. Jacobsen, Robert. Analysis of Structures for Packet Sorting Networks. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 163. Cambridge, Ma., July 1978.

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

16. Misunas, David P. Report on the Second Workshop on Data Flow Computer and Program Organization. M.I.T., Laboratory for Computer Science, LCS/TM-136. Cambridge, Ma., June 1979.

17. Vishniac, Ephraim. A Processor Module for Data Flow Computer Development. M.I.T., Laboratory for Computer Science, Computation Structures Group, Memo 176. Cambridge, Ma., May 1979.

Accepted for Publication

1. Arvind, and Bryant, Randal E. "Parallel Computers for Partial Differential Equation Simulation." To be published in Proceedings of Scientific Computer Information Exchange Meeting.

2. Brock, J. Dean, and Montz, Lynn B. "Translation and Optimization of Data Flow Programs." To be published in Proceedings of the 1979 International Conference on Parallel Processing.

3. Bryant, Randal E. "Simulation on a Distributed System." To be published in Proceedings of the First International Conference on Distributed Computing Systems.

4. Dennis, Jack B. "The Varieties of Data Flow Computers." To be published in Proceedings of the First International Conference on Distributed Computing Systems.

5. Dennis, Jack B., and Weng, Kung-Song. "An Abstract Implementation for Concurrent Computation With Streams." To be published in Proceedings of the 1979 International Conference on Parallel Processing.

6. Dennis, Jack B.; Leung, Clement K. C.; and Misunas, David P. "A Highly Parallel Processor Using a Data Flow Machine Language." To be published in IEEE Transactions on Computers.

7. Leung, Clement K. C. "ADL -- An Architecture Description Language for Packet Communication Systems." To be published in Proceedings of the 1979 International Symposium on Computer Hardware Description Languages and Their Applications.

Theses Completed

1. Borkin, Sheldon A. Equivalence Properties of Semantic Data Models for Database Systems. Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1979.

2. Brock, Jarvis D. Operational Semantics of a Data Flow Language. S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, October 1978.

3. Hong, Peter. "Analysis of Buffering Requirements in a Data Flow Processor." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

4. Kosinski, Paul. Denotational Semantics of Determinate and Non-Determinate Data Flow Programs. Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

5. Redford, John L. "A Design for a Routing Module." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, January 1979.

6. Ressler, Paul. "Simulation of a Highly Parallel Processor." unpublished S.B. Thesis, January 1979.

7. Tetrick, Scott. "An Instruction Cell Block Design for a Data Flow Computer." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

8. Vishniac, Ephraim M. A Processor Module for Data Flow Computer Development. S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

9. Weng, Kung-Song. An Abstract Implementation for a Generalized Data Flow Language. Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

Theses in Progress

1. Isaman, David L. "Systems of Data Structuring Operations for Parallel Processors." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1979.

2. Leung, Clement. "Fault Tolerance in Packet Communication Computer Architecture." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, March 1980.

3. Montz, Lynn B. "Safety and Optimization Transformations for Data Flow Programs." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, February 1980.

Talks

1. Ackerman, William B. "History Functions Cannot Describe All Packet Communication Systems." Second Workshop on Data Flow Computer and Program Organization, M.I.T., Dedham, Ma., July 1978.

2. Ackerman, William B. "A Structure Processing Facility for Data Flow Computers." International Conference on Parallel Processing, August 1978.

3. Ackerman, William B. "Data Flow Languages." National Computer Conference, New York, N. Y., June 5, 1979.

4. Arvind. "Irvine Dataflow Language." Oxford University, Oxford, U. K., August 8, 1978.

5. Arvind. "An Asynchronous Programming Language and Computing Machine." Distributed Computing Panel, Rutherford Laboratory, Oxfordshire, U. K., August 9, 1978; University of Newcastle Upon Tyne, Newcastle Upon Tyne, U. K., August 14, 1978; University of Minnesota, Minneapolis, Minn., November 8, 1978; Michigan State University, East Lansing, Mich., November 9, 1978.

6. Arvind. "Nondeterministic Programming in Dataflow." Rutherford Laboratory, Oxfordshire, U. K., August 9, 1978; University of Newcastle Upon Tyne, Newcastle Upon Tyne, U. K., August 15, 1978.

7. Arvind. "Computing With Streams." Laboratory for Computer Science, M.I.T., Cambridge, Ma., November 14, 1978; Dataflow Workshop, CERT, Toulouse, France, February 13, 1979.

8. Arvind. "A Brief Introduction to Dataflow." National Computer Conference, New York, N. Y., June 5, 1979.

9. Arvind. "Mapping a Hydrodynamics and Heat Conduction Problem on a Dataflow Computer." National Computer Conference, New York, N. Y., June 5, 1979.

10. Borkin, Sheldon A. "Data Model Equivalence." Fourth International Conference on Very Large Data Bases, W. Berlin, Germany, September 1978.

11. Boughton, George A. "Structure of Routing Networks." Second Workshop on Data Flow Computer and Program Organization. M.I.T., Dedham, Ma., July 12, 1978.

12. Bryant, Randal E. "Analytical Models for Interconnection Networks." Second Workshop on Data Flow Computer and Program Organization. M.I.T., Dedham, Ma., July 12, 1978.

13. Dennis, Jack B. "Status and Goals of Data Flow Research at M.I.T." Second Workshop on Data Flow Computer and Program Organization. M.I.T., Dedham, Ma., July 10, 1978.

14. Dennis, Jack B. "A User Language for Data Flow Computation." Second Workshop on Data Flow Computer and Program Organization. M.I.T., Dedham, Ma., July 10, 1978.

15. Dennis, Jack B. "Data Flow Computer Architecture." Burroughs Corp., Paoli, Pa., July 17, 1978; University of Toronto, Toronto, Canada, October 25, 1978; University of Texas, Austin, Tx., December 11, 1978; Avionics Laboratory, Wright-Patterson Air Force Base, Ohio, December 13, 1978; Carnegie Mellon University, Computer Science Department, December 14, 1978.

16. Dennis, Jack B. "The Programming Language VAL." University of Waterloo, Computer Science Department, Waterloo, Ontario, Canada, October 26, 1978.

17. Dennis, Jack B. "Research Directions in Computer Architecture." University of Waterloo, Computer Science Club, Waterloo, Ontario, Canada, October 26, 1978.

18. Dennis, Jack B. "Data Flow Computing: Architecture and Issues." Prime Computer, Inc., Framingham, Ma., October 31, 1978.

19. Dennis, Jack B. "The Varieties of Data Flow Computers." National Computer Conference, New York, N. Y., June 5, 1979.

20. Hirschman, David. "Translation of a Large Fortran PDE Code into Data Flow Form." Second Workshop on Data Flow Computer and Program Organization, M.I.T., Dedham, Ma., July 11, 1978.

21. Leung, Clement K. "Fault Tolerance in Packet Communication Architecture." Second Workshop on Data Flow Computer and Program Organization, M.I.T., Dedham, Ma., July 12, 1978.

22. Montz, Lynn. "Safety and Optimization Transformations for Data Flow Programs." Second Workshop on Data Flow Computer and Program Organization, M.I.T., Dedham, Ma., July 11, 1978.

23. Weng, Kung-Song. "An Approach to Data Flow Support for Procedure Invocation and Streams." Second Workshop on Data Flow Computer and Program Organization, M.I.T., Dedham, Ma., July 11, 1978.