LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Computational Structures
Progress Report for 1983-84

Computation Structures Group Memo 246

February 28, 1985

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

COMPUTATION STRUCTURES

Academic Staff

J. B. Dennis

Visitor

G. Lindstrom

Research Staff

W. B. Ackerman
C. K. C. Leung
W. Y-P. Lim

Graduate Students

N. B. Bauman
G. A. Boughton
J. D. Brock
T. A. Chu
G-R. Gao
B. Guharoy
T. R. Hegg
S. Jagannathan
K. B. Theobald
T. S. Wanuga

Undergraduate Students

J. Holderle
B. Kartz
D. Kravitz
B. Kuszmaul
B. Lim
E. Lyons
D. Marcovitz
S. Markowitz
T. Tran

Support Staff

P. Sedell

Introduction

The work of the Computation Structures Group concerns new concepts of the basic structure of computer systems, and therefore addresses issues ranging from hardware design methodology, through the consistent specification of machine architecture and user language, to the evaluation of applications for execution on proposed system architectures. Two system designs are currently being pursued: the static data flow architecture for large scale scientific computation; and the VIM Project which is exploring the application of data flow principles to a general purpose system architecture. In the development of the static architecture, progress has been made in the architectural design of a practical data flow processing element, in the experimental design of key LSI components, in performance studies, and in the techniques of program transformation essential to an effective compiler for VAL (the functional programming developed by the group for use with data flow computers). Work on the VIM Project has centered on the design of representations for the structured data types of VIMVAL (the user language for VIM), the philosophy of type inference and type checking to be incorporated in VIM, and study of the problems of maintaining data integrity through automatic backup. The group is also working on an advanced methodology for the design of self-timed logic circuits in continuation of its previous work in this area.

Data Flow Processing Element

The principal effort of the Computation Structures Group is work toward construction of a demonstration data flow machine for high-speed numerical computations. The goal is to demonstrate a machine that can achieve better than 100 megaflops of performance in executing useful programs compiled from VAL. The work is guided by our experience in the analysis of benchmark programs in application areas from weather modeling to plasma dynamics.

The design problem is simplified by the recent availability of a commercial, high performance floating point chip set comprising a multiplier and an adder. Because of their

ability to pipeline successive operations, they are well suited to to the construction of a data flow processor.

The envisioned machine has 64 processing elements, each built around a pair of floating point chips and capable of at least five megaflops of performance. We estimate that each processing element will hold several thousand data flow instructions and will include a large local memory for holding the arrays of values that make up the data base of an application. We expect the machine will be able to achieve its full potential performance of 320 megaflops for many application codes. Several proposals for the organization of the processing element have been developed. These are being compared and evaluated, and a choice of one design for detailed development will be made during the next year.

Program Transformation

The success of a practical data flow computer for high performance scientific computation hinges on the ability of a compiler to produce effective data flow programs from high-level source programs. The basic problem is to create target program structures that adapt the degree of parallelism exposed in the source program to the storage capacity of the machine. To do this, the compiler must restructure the program to make the trade-off between time (parallelism), and space (memory) that permits full utilization of the machine. The Computation Structures Group is exploring two avenues toward this goal--a general approach based on program transformations that apply to any program written in VAL, and a second approach that generates a *pipe-structured* data flow program for any of a certain class of VAL programs.

In his recent doctoral thesis [2], William Ackerman studied the translation and optimization techniques for VAL programs that deal with arrays. The principal transformation technique proposed for achieving high execution speed on a static data flow computer is the *spatial interleaving* of arrays and the *unfolding* of the iteration loops that manipulate them. This technique allows the transformed program to be distributed over many processing elements, permitting parallel operation with minimal communication

among the parts. This technique works for the *forall* expression of the VAL language as well as the normal sequential iteration loop, and it works for nested loops and multidimensional arrays. It should yield good results over a wide range of machine sizes and problem sizes.

Gao Guang-Rong has studied a class of VAL programs in which the body of the program consists of several blocks of code interconnected without cycles. Each block defines one or more array values in terms of its inputs, and has the form of an iteration or a *forall* expression. Gao has shown that such programs can be transformed algorithmically into data flow machine code in which the data is pipelined through the instructions at the maximum rate permitted by the architecture. This approach permits calculation of the instruction and data storage required, and therefore the right choice of space versus time may be made to utilize as much of the machine's capacity as possible. The method used to construct pipelined code by augmenting data flow programs with buffering is treated in earlier work by Gao [8].

Incorporating these techniques into a practical compiler is a challenging task. Yet the functional (applicative) nature of the VAL programming language makes the task much easier than would be the case for a more conventional language such as Fortran. Construction of a program-transforming compiler for real-world application programs will be a major project over the next few years.

Logic Design Methodology

The Group has contributed to the study of asynchronous design of digital logic systems for many years [4], and specifically to design techniques based on self-timed concepts of logic design [9]. At one time it appeared that self-timed techniques already developed might be competitive in the context of VLSI design [9]. However, our experimental design of a two-by-two router device in LSI using standard self-timed circuit elements showed that this approach is far from achieving competitive component density.

Tam-Anh Chu has conceived a design method that exploits silicon layout and self-timed principles together to achieve designs that are competitive with conventional techniques. The methodology uses a novel system organization. In contrast to the usual data-path/controller partition of synchronous systems, a system is organized into *data-path*, *state-machines*, and *distributed control structures*. Self-timed data-path circuits are built using logic gates, pass-transistors, and matched timing delays. Such an approach yields layouts of almost the same area as for synchronous construction, and is possible because in VLSI systems, delays of data-path components can be easily matched by using identical geometry. Also, data-path components have their own controllers for inter-module timing synchronization. State-machines and distributed control structures are used together to form the control part of the system. State-machines test predicates only to guide control flow in the distributed control, and are simple and fast.

A self-timed router chip [6] has been designed using this approach. The router is a packet switching device with two input and two output ports. It decodes address bits of byte-serial input packets and routes them to designated output ports by setting up and maintaining a link between ports throughout the duration of a packet transmission. Packets going to different output ports can be processed concurrently, and any contention for output ports is resolved using arbiters.

The methodology uses a graph model called the Signal Transition Graph for the direct synthesis of self-timed circuits. A Signal Transition Graph is a directed graph where vertices represent signal transitions at circuit nodes, and arcs specify static and dynamic relationships between pairs of transitions. Additional constraints are included in the graph to permit determination of the circuit (an interconnection of components), and the logic functions of the components. Two self-timed chips have been successfully designed using this graph model--a router chip and a ring buffer chip.

Integrated Circuit Design and Fabrication

Since the use of custom integrated circuit devices is essential to achieving competitive performance in a data flow processor, we have developed several experimental chips using CAD tools available at MIT, and the DARPA MOSIS fabrication service. The devices chosen for experiment reflect key requirements in the construction of practical data flow computers. One is the two-by-two router which is a building block for packet switched routing networks. The second is a simple data flow processing element with 8-bit data words that has given us experience in implementing all the essential mechanisms of a full-scale data flow processor.

The first device constructed [5] was a scaled down router with a two-bit data-path. It was fabricated in 1983 using a 5-micron NMOS process. We have obtained eight chips from MOSIS, all of which were found to be fully operational at the relatively slow rate of around 0.7MHz. This router was designed using self-timed circuits with extremely conservative expectation of process variations. Dual-rail coding was used in the data-path and control circuits to ensure fully speed-independent operation of all components. Also, a substantial amount of circuitry was included for test purposes, and this partly degraded the performance of the router. In retrospect, this project was successful in that it allowed verification of the design approach and produced valuable feedback.

Our experience with this design inspired development of the new design methodology described above. Application of the methodology to the two-by-two router yielded a device [6] with a full 9-bit data-path and buffer storage for eight bytes at each input port. It has been fabricated through MOSIS using a 3-micron CMOS process, and its speed is estimated to be 5MHz. In this version, control circuits are speed-independent and data-path components are timed by means of matched delays. All modules are designed so that times for the reset phase of the four-cycle signaling protocol are reduced to a minimum, yielding a significant overall speed improvement. The layout of the data-path in this circuit is similar to a synchronous one. We believe the layout of the control is superior to that of

the synchronous design, as the distributed control structures uses less circuitry than a central controller, and they may be laid out to the same pitch as the data-path.

The simple data flow processing element was an exploratory design developed as a course project in 1983. It has been redesigned and submitted to MOSIS for fabrication. This chip was implemented in bulk CMOS with 3 micron features. When combined with a commercial 2K by 8 RAM chip it forms a simple processing element that can handle the sequencing of 64 data flow instructions. Most of the chip area is occupied by an eight-bit arithmetic-logic unit/register array, and several PLA structures that implement the control functions. The unique element is a 64-bit *enable memory* having an integral priority encoder that implements the data flow instruction sequencing rules. In a full-scale data flow processor, the size of the enable memory will require that it be fabricated as one or more dedicated chips. Such a specialized device will be the next candidate for experimental design.

Performance Studies

Being very different from conventional computer systems, data flow computers require new approaches to performance evaluation. One important aspect is the manner in which the instructions of a data flow program are assigned to the processing elements of the machine so that full performance may be achieved. This aspect is addressed primarily as a problem of program structure, and compiler analysis and transformation of programs. The other aspect of performance of the static data flow multiprocessor is the manner in which the load placed on the routing network by the processing elements interacts with the protocol of the routers to determine the rate at which data flows among the processing elements.

In most of our work we have focussed on the packet routing network as an independent subsystem. In his doctoral thesis [3] G. Andrew Boughton examines the design of high throughput packet switched networks for interconnecting the modules of a large digital system such as the processing elements of a data flow computer. The design of such

networks is studied under two different sets of assumptions about the implementation technology. The first set corresponds roughly to present technology where only a small number of network nodes can be placed on a single integrated circuit chip. The second set corresponds to future VLSI technology where a large number of network nodes can be placed on a single chip.

Under the first set of assumptions, network structures based on the indirect n-cube topology are studied to determine the factors limiting the performance of very large networks. For this purpose the load on the network is assumed to be generated by independent packet sources at each input port with uniformly distributed packet destinations. For such sources, the strongest constraint examined still allows the throughput of the network to grow linearly with the number of network inputs. However, we show that conditions can arise in the operation of moderately large networks (around a thousand inputs) such that some network inputs accept packets at less than half the average input rate for a long period of time. The design of networks where the sources generate a nonuniform distribution of packet destinations is examined briefly.

For network implementations where many nodes are built into a VLSI chip, we derive a minimum wire cost proportional to the square of the number of network inputs, shown for networks capable of high performance for certain uniform patterns of communication. Networks are presented that come close to supporting the desired level of performance using the specified amount of wire. Networks for other patterns of communication are also briefly discussed.

Research is beginning on how the routing network influences performance of the static data flow machine. The effects of packet traffic flow patterns generated by the program workload on the throughput of the routing network (as well as the rest of the machine) are being examined. From the results of this analysis, possible methods for improving the performance of the routing network are being considered. Initially, the structure of the

routing network is being restricted to the indirect binary n-cube, and the program workload is being restricted to the class of pipe-structured programs.

General Purpose Data Flow Computing: The VIM Project

The Computation Structures Group is developing an experimental advanced general purpose computing system based on the principles of data flow computation and functional programming and using the model of computation proposed by Dennis [7]. The prototype for such a system is the VAL Interpretive Machine or VIM. The high-level functional programming language chosen for this project is VIMVAL, a revision and extension of VAL [1]. Our current work on VIM covers three areas: the development of a comprehensive memory system for supporting VIMVAL data structures; development of a simple but general type system for VIMVAL using type inference; and the development of specialized backup and recovery procedures for VIM.

VIM Structures

The physical storage system of VIM consists of a semiconductor main store and a disk. A common virtual address space is provided to all users to facilitate the sharing of programs and data. Ideally, the only information brought from the disk into the main store should be the logical units of information--such as records and arrays--which are referenced by the computation. A large page size is undesirable because inefficient memory use results when independent logical units are packed into the same page. In conventional systems a small page size is troublesome because the large page traffic cannot be handled efficiently. In particular, the overlapped execution of several page accesses for one computational process is generally impractical.

In the case of VIM, the concurrency of the data flow program execution model makes the overlapped handling of page accesses much easier to consider. Therefore we are examining the issues involved in implementing a paging mechanism for VIM with a small page size. The unit of transfer between the main store and the disk is called a *chunk* and its size has been set at 32 words for our present experimental design.

Bhaskar Guharoy has designed representations for the principal structured data types of VIM-- and records--to permit efficient operation of the storage management system. A large array or nested record structure is represented as a tree of chunks. In this representation, data structures may share common substructures, and the basic operations of accessing an element or creating an altered version of a structure can be done in $\log(n)$ steps for a structure of n elements. Furthermore, a full (modified) copy of a data structure may be made in $\log(n)$ time.

The design of VIMVAL and its implementation is such that circular structures are never created during system operation. This acyclic property of the data structures permits use of a reference-count based storage reclamation mechanism.

The VIM Type System

A type system has been developed for the revised version of the VAL programming language (VIMVAL) in a bachelor's thesis completed by Bradley Kuszmaul. The type system is designed so that users of VIMVAL may omit type declarations from their programs whenever the type of an expression can be determined by the compiler. The system combines several ideas that have been found to be worthwhile in advanced language designs. To start with, VIMVAL includes higher order functions, that is, functions are "first class objects" and may be passed as values and built into data structures. The type system allows programs to be written with incomplete type specifications, and the type checker infers the types of expressions from their context. The types of some expressions may remain undetermined even after program compilation. This allows the binding of types to be deferred until the module is linked to other modules to create an executable program. At that point all types must be determined, for we wish to exclude the possibility of discovery of type errors during program execution. This aspect of the type system allows program modules to be "polymorphic"--a module may perform analogous operations on different types, according to the modules linked to it.

The thesis develops a theory of types which applies to a large class of programming

languages, including VIMVAL. The notion of type is defined in terms of regular sets, which describe sequences of legal operations on a value of a given type. Recursive types allow infinite sequences of legal operations, so the definition of types allows infinitely long sequences to be elements of the regular sets describing a type.

A program in VIMVAL can be translated into a data flow graph and the local constraints the program places on the types of its expressions can be described in terms of the class of regular sets which satisfy an operator's type requirements. Type correctness can be defined in terms of the number of regular sets which satisfy every operator's type requirements: a program is type correct if there is exactly one regular set satisfying all the type requirements of all the operators in a program. Type correctness is well defined in terms of mathematical set operations. Type correctness has been shown to be decidable, and an efficient type checking algorithm for VIMVAL has been developed on the basis of this theory.

Backup and Recovery Issues in VIM

We wish to provide a backup and recovery system for VIM that will guarantee the security of all online data. We envision that incremental backup information will be maintained on some stable storage device--stable disk--with older backup information being held on tape storage. The objective is that, following loss of data due to hardware failure, full recovery of the system to the situation prevailing just prior to the fault is possible. That is, users will be able to continue as if no fault had occurred except for the loss of a few characters from terminal input/output buffers.

An interesting approach is made possible by the applicative nature of the VIMVAL language: since reexecution of any procedure that is a function is guaranteed to produce the same results, even in the presence of arbitrary concurrent computations, it is not necessary to back up intermediate states of function evaluation; only the arguments of the invocation need be saved. This concept becomes more intricate as one considers data structures and stream-oriented computation. Of course, *guardians*, the mechanism provided in VIM for implementing nondeterminate computations, must be treated carefully. These issues are the subject of master's thesis research in progress by Suresh Jagannathan.

References

1. Ackerman, W. B. and Dennis, J. B. VAL -- A Value - Oriented Algorithmic Language: Preliminary Reference Manual. Technical Report TR-218, Laboratory for Computer Science, MIT, Cambridge, MA 02139, June, 1979.

2. Ackerman, W. B. *Efficient Implementation of Applicative Languages.* Ph.D. Th., EECS, April 1984.

3. Boughton, G. A. *Routing Networks for Packet Communication Systems.* Ph.D. Th., Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139, 1984.

4. Bryant, R. E. Report on the Workshop on Self-timed Systems. Tech Memo 166, Laboratory for Computer Science, MIT, Cambridge, MA 02139, May, 1980.

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

6. Chu, T-A. Design of a Self-Timed Router using Signal Transition Graphs and VLSI Techniques. In preparation.

7. Dennis, J. B. Data Should Not Change: A Model for a Computer System. CSG Memo 209, Laboratory for Computer Science, MIT, Cambridge, MA 02139, July, 1981. Submitted for publication..

8. Gao, G.-R. An Implementation Scheme for Array Operations on Static Data Flow Computer. Master Th., LCS, Cambridge, MA,June 1982.

9. Seitz, C. System Timing. In Chapter 7 of Introduction to VLSI Systems, Reading, MA, 1980.

Publications

1) Dennis, J.B., "Data Flow Ideas for Supercomputers," Proceedings of CompCon '84/28th IEEE Soc. Inter. Conf., February-March, 1984.

2) Dennis, J.B. and Gao G-R., "Maximum Pipelining of Array Operations on Static Data Flow Machine," 1983 International Conference on Parallel Processing, Bel Air, MI, August 1983.

3) Dennis, J.B., Lim, W. Y-P., and W. B. Ackerman, "The M.I.T. Data Flow Engineering Model," Proceedings of IFIP 9th World Congress, Paris, France, September 1983.

Theses Completed

1) Ackerman, W.B., "Efficient Implementation of Applicative Languages," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, April 1984.

2) Kuszmaul, Bradley, "Type-checking in VIM-VAL," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, May 1984.

3) Brock, J. Dean, "Formal Model of Non-determinate Data Flow Computation," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, August 1983.

Theses in Progress

1) Boughton, G.A. "Routing Networks for Packet Communication Systems," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

2) Chu, T-A. "A Design Methodology for Self-timed VLSI Systems," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1986.

3) Gao, G-R. "A Pipelined Code Mapping Scheme for Static Data Flow Computers," Ph.d. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1985.

4) Guharoy, B. "Memory Management in a Static Data flow Computer System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1985.

5) Jagannathan, S. "Guaranteeing Data Security in a Dynamic Data Flow Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1985.

6) Theobald, K. "Adding Fault-Tolerance to a Static Data Flow Supercomputer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1985.

7) Wanuga, T. Routing Performance in a Static Data Flow Computer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1985.

Talks

1) Dennis, J.B., "Design Issues in Functional/Applicative Programming Languages," MIT Data Flow Workshop, Dedham, MA, July 1983.

2) Dennis, J.B., "Non-determinate Computation Using Streams and Guardians", MIT Data Flow Workshop, Dedham, MA, July 1983.

3) Dennis, J.B., "Type-Checking and Inference for VimVal," MIT Data Flow Workshop, Dedham, MA, July 1983.

4) Dennis, J.B., "Architectural Abstraction," Summer Study on Methods for Improving Software Quality and Life Cycle Costs," NAS Summer Study Center, Woods Hole, MA, July 1983.

5) Dennis, J.B., "Data Flow Ideas for Supercomputers," Conference on the Frontiers of Supercomputers," Los Alamos, August 1983.

6) Dennis, J.B., "Language-Based Design of Computer Systems," for the short course: "Software-Oriented Computer Architecture," Boston, MA, November 1983.

7) Dennis, J.B., "The TX-O at MIT," The Computer Museum, Marlboro, MA, November 1983.

8) Dennis, J.B., "V*im*: An Experimental Computer System Supporting Functional Programming," Conference on High-level Language Computer Architecture, Los Angeles, CA, May 1984.

9) Dennis, J.B., Remarks in accepting the 1984 ACM-IEEE Eckert-Mauchly Award, Ann Arbor, MI, June 1984.

10) Gao, G-R., "Maximum Pipelining of Array Operations on Static Data Flow Computers," 1983 International Proceedings on Parallel Processing, Bel Air, MI, August 1984.

11) Lim, W. Y-P. "The M.I.T. Data Flow Engineering Model," IFIP Congress, 9th World Congress, Paris, France, September 1983.