MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group Memo No. 50

Proposed Research in Computation Structures*

Jack B. Dennis

April 1970

* Part of the Project MAC continuation proposal to the Advanced Research Projects Agency, January 1970.

Introduction

The Computation Structures Group is concerned with advanced concepts in the organization of general-purpose computer systems -- and especially with the influence of the demands of programming and style of computer usage on computer architecture. The research carried out by the group is based on this premise:*

"It is generally agreed that the cost of software development is now the dominant expense in the application of computers. The cost of computing hardware appears destined to diminish indefinitely. Yet the cost of programming is bound to rise as it directly represents human effort."

The implication is clear: Any innovation in computer architecture that can effect a reduction in the effort required to create working programs is worth very serious consideration, even if it appears to reduce the "efficiency" with which computations are performed.

Our (the Computation Structures Group's) approach to innovation in computer system organization is through defining with greater care the function performed by a computer system. Any computer system defines a programming language; the primitive operations of the language correspond to the instruction types implemented by the processing hardware, the commands that may be given to the operating system by running programs, and the functions implemented by a basic set of library subroutines. In the design of existing computer systems, negligible thought has been given to the language it defines -- the language is the result of a multitude of decisions made by independent groups, and very often with little rational thought.

* Quotations are from publications of the Computation Structures Group.

We propose to work toward the design of an advanced computer system by first specifying the language it is to implement, and then developing a computer architecture that can provide an effective realization of the language. For lack of a better term, we will call the language, whose definition we seek, the <u>basic language</u>.

<u>Sharing of Programs and Data</u>

To further understand the characteristics we desire in the basic language, we briefly review our hypotheses regarding the nature of future computer systems. We assume that interaction with a large, shared computer installation, or network of computer installations, from a personal console will be the normal mode of engaging computing and information services. Multiprocessing and multiprogramming will be used so resources of a computer installation will be multiplexed among many independently written programs. The major portion of information required to serve users will reside in on-line files.

We expect an increasing demand from computer users for use of programs and data bases prepared at other computer installations. When it becomes possible to achieve sharing of programs and data bases through the facilities of a network of computer installations, the concept of the "computer utility" will have been realized. Implementing the computer utility requires introduction of the concepts of ownership of information and access control into the design of computer systems. The requirements that the controlled sharing of procedures and data places on the structuring of information within a computer installation (or in a network) have been studied by the Computation Structures Group. The results of this study will be incorporated into the specification of the basic language.

Yet access to a program or data base is not sufficient to make it useful in a context different from the context assumed by its owner -- problems of compatibility must be overcome. A program cannot produce identical results at different installations unless all primitive operations used in the program have equivalent effects at each installation. Procedures expressed in different programming languages cannot be used together unless they employ identical representations of common data. If a data base is to be used at different installations, then the procedures that operate on the data base must be expressed in programming languages that employ equivalent representations for all data types occuring in the shared data base.

We believe a truly effective solution to the compatibility problems that plague attempts to share information is to be achieved only through adoption of a common intermediate representation for those procedures and data that are to be shared. We intend that the basic language be a practical common intermediate form for programs and data specified in a significant range of popular programming languages.

Programming Generality

"A most important means of creating a program is to construct it from existing programs. Very often the exact program for an application does not exist, but major components of it have been expressed as programs, very likely in different languages.

"Programming generality is the property of a computer system that enables users to combine existing program modules, independently expressed in different languages."

Programming generality is a rare property in contemporary computer systems: It is generally not possible to combine independently written programs without making internal changes. The exceptions are when the component programs are small and do not employ dynamic storage allocation (e.g., basic library subroutines), or when the authors have expended much effort to ensure that a particular set of modular programs can be used together.

We believe the attainment of programming generality has far-reaching implications for the architecture of computer systems. The following summary of a relevant argument is taken from the abstract of a paper presented to the IFIP Congress 68:

"To exhibit programming generality, a computer system must permit a program module to 1) create information structures of arbitrary extent; 2) call on procedures with unknown requirements for storage and information structures; and 3) transmit information structures of arbitrary complexity to a called procedure. These requirements imply that the hardware or operating system must make storage allocation decisions and, hence, that location-independent addressing must be used. A small unit of storage allocation is necessary to prevent waste of valuable storage capacity and unnecessary information transfer. Exploitation of detailed parallelism [in computations] is then essential to fully utilize the processing hardware."

Programming generality is, however, just one explanation for our interest in parallelism.

Parallelism

Many changes in the field of computing and information processing point to the increasing importance of concepts of computer system organization that permit many parts of a computation to be carried forward concurrently. Designers have reached the limit of their ability to devise more powerful processing units by adding complexity to hardware that interprets a sequential representation of a process. The emergence of large-scale integrated circuits suggests that improvements in performance will be achieved through parallelism rather than increased speed of logic circuits. To these arguments we add another arising from our interest in systems that offer programming generality: To exhibit programming generality and at the same time to be efficient, a computer system must multiplex its resources among many computations so resources not required by one computation are not left idle. Exploitation of parallelism is required for efficient utilization of processing and memory units if programming generality is to be obtained in a system having several levels of physical memory.

Proposals advanced elsewhere for highly parallel computer systems have neglected the question of programming generality. The Computation Structures Group has been studying possible machine organizations that are highly parallel, yet have the necessary features to achieve programming generality. Our present concept of such a computer is briefly this:

> " The machine has two major parts -- the <u>memory system</u> and the <u>processing hardware</u>. The memory system has, say, three levels, M_A, M_B, M_C ... . In each level addressing is associative using a key consisting of a pointer [to an information structure] and a name byte [that selects one of its components]. ... Items are brought into higher levels of memory on demand ... . The organization of the processing hardware is intended to permit extensive sharing of multiple specialized units by many computations ... using a 'service on demand' principle of control."

This concept of computer organization is consistent with our present thoughts regarding specification of the basic language.

<u>Research Program</u>

Future research in the Computation Structures Group will fall in two principal areas: One is the study of theoretical and practical issues concerning specification of the basic language. The other is the study of problems relating to implementation of the basic language in the form of a highly parallel computer system. The near-term activities projected for these two branches of research are discussed in the following paragraphs.

<u>Basic Language</u>

The foundations for the specification of the basic language are provided by past research, within and outside Project MAC, on the representation of parallelism in computation. Of particular importance is the work of Rodriguez, who introduced the concept of "program graph" and clarified issues relating to parallel programs containing decisions and iteration. We have extended Rodriguez's work and used a variation on his program graphs in teaching an undergraduate subject on implementation of computations.

For the representation of structured information, we have defined a graph-theoretic model that is a generalization of trees. We have found that our model is nearly identical to the abstract "objects" used by the IBM Vienna Research Group for their definition of PL/1. It is also a simple extension of Landin's abstraction of structure often used in connection with the lambda calculus model of computer programs.

Issues that must be resolved to formulate a complete specification of the basic language include both theoretical and practical matters. The most important theoretical questions relate to representing parallel computations on structured information, modeling co-routines, and representing parallel computations of the form "apply procedure P for each element x of the set X." Other issues concern the specific choice of primitive operations and data types, and the design of access control and debugging features of the language.

In parallel with the development of specifications for the basic language, we will study the problem of specifying translators from important source programming languages into the basic language.

The work toward a specification of the basic language will be supported partly by a grant from the National Science Foundation and partly by the main Project MAC funding toward which this proposal is directed. In this effort we expect to work closely with the Programming Linguistics Group, for their research in formal semantics and extensible languages is obviously relevant.

Associative Memory Systems

We explained earlier how progress toward programming generality requires that the machine level representation of procedures must encode references to information in the form of location-independent tokens. That is, the tokens or identifiers used to refer to variables and information structures must not have any implication respecting where in memory the variables and information structures are stored.

In current systems (such as Multics) the efficiency with which location-independent addressing is implemented is severely limited by the fact that different identifiers are used to locate information within the different levels of memory. Software procedures must be invoked to make conversions of identifiers whenever information is moved between levels of memory. If a uniform scheme of identification is adopted for all memory levels, then it is possible to design hardware structures that perform all parts of the memory allocation task. (This has been done in the "cache" memory of the IBM Model 360/85 system.) In such systems, the higher levels (least capacity, fastest access) of the memory hierarchy must be accessed associatively.

For this reason, we are studying the design, organization, and performance of hierarchial associative memories. We are interested in associative memories that provide for concurrent servicing of many simultaneous requests for retrieval of information. In this context, shift registers, delay lines, and magnetic drums are attractive, and we are studying ways in which these serial devices may be used in associative memory systems having a high capacity for serving concurrent requests.

Asynchronous Modular Systems

We believe that successful construction of a highly parallel computer, such as we envision, requires the use of asynchronous communication among its parts. For several years the Computation Structures Group has been studying the representation, design, and implementation of asynchronous, modular switching systems. This work is closely related to the work on "macromodular systems" led by W. A. Clark at Washington University with ARPA support. Whereas their work has the practical objective of making it easy for individuals to construct specialized digital systems, current research in the Computation Structures Group is concerned with the theory of schemes for representing such systems. Our most recent success has been the conception of "coordination nets" -- a scheme for representing the coordination requirements of independently timed activities. We have shown how to obtain a design for an asynchronous modular system that implements the coordination of events specified by an arbitrary coordination net. Situations requiring hardware coordination of concurrent activities will be common in highly parallel machines of the future. We owe much to Anatol W. Holt as we have made very profitable use of his ideas, especially his "Petri nets", in our research.

We expect to carry further our theoretical studies of asynchronous systems, and to begin experimental work to evaluate the applicability of our concepts to the design of large-scale systems. We hope to develop a theory which will characterize formally those interconnections of logic gates that define deterministic asynchronous hardware modules. (We already know very natural communication constraints under which systems formed from deterministic subsystems may be guaranteed to be deterministic). We anticipate that these and related theoretical achievements will eventually replace the classical switching theory (due to Huffman and Caldwell) as the foundation for the design of asynchronous digital systems.

We wish to set up a modest "laboratory for asynchronous switching systems" for experimentation with applications of our theory to design problems. We will design and construct experimental versions of basic control modules, demonstrate their use in typical digital subsystems, illustrate the implementation of coordination nets, and explore the design of time-dependent computing modules.