AN EXTENSIBLE ARCHITECTURE FOR DATA FLOW PROCESSING

Bezalel Gavish and Harvey Koch
Graduate School of Management
University of Rochester
Rochester, New York 14627

Abstract

In this research, we propose a general model for computer architectures. This model allows us to integrate some of the existing system architecture approaches for increasing computer capacity and indicates new possibilities for improving the cost/performance ratio of future computer systems. The model leads to the development of general purpose data flow machines and to different degrees of sophistication within these machines. Our fundamental approach is to separate the CPU into elementary components. This allows us to make a distinction between components for control and components for performing various processing functions, thus yielding a high degree of resource sharing.

Introduction

One of the major characteristics of the computer industry in the last 20 years is the continuous need and demand for faster computers and higher capacities of computation (measured, for example, by MIP's). In contrast to the past, where the driving force was what may be classified as the scientific community, in recent years we are observing a dramatic increase in the demand for computation power by commercial users (as evidenced, for example, by the large number of orders for the IBM 30X family). This demand may be mainly attributed to the development of large scale computer communication networks, in which the computer is required to maintain and drive networks containing thousands of terminals generating up to a hundred transactions per second directed towards very large data bases. Examples of applications and organizations in which such on-line services supply enough economic incentives for integration and for the development of large on-line services are air reservation systems, the banking industry, on-line retailing, large on-line distribution and marketing systems in the auto industry, automatic distribution and control of freight cars in the railway industry, on-line inventory management systems in large organizations, on-line medical data bases and computer assisted diagnosis.

Four basic approaches have been developed over time for increasing the capacity of computer systems. One approach is increasing the speed rate of the elementary components of the central processing unit by reducing the switching time of logic devices and circuits and shortening the interconnection lines. In the past twenty years, tremendous progress has been made in device and circuit technology and in miniaturization techniques. However, physical and technological boundaries (see Keyes [9]) together with the increased investments needed for those technologies limit the advancements in this direction.

A second means of increasing the computer capacity can be obtained through advancements in computer architecture. Methods such as pipelining, look-ahead and parallelism have been used to increase computer capacity by allowing execution of multiple instructions in parallel. However, it seems that in this area some stabilization has been reached and no major breakthroughs have occurred recently.

The third approach, which has been proposed for over 15 years, is multiprocessing, i.e., having more than one processor in the computer system under centralized control. By directing the workload to various processors, it is possible to increase the execution rate. It seems that despite all the theoretical advantages of the multiprocessing systems, no general purpose multiprocessing system has been developed which is acceptable by the user community and which actually generates those benefits (see Enslow [7]). Moreover, the transition from a uniprocessor to a multiprocessor environment is not a simple process, requiring in many cases reprogramming and redesigning of applications in order to obtain some of the advantages promised by this approach.

In the last five years, a fourth approach has emerged for data processing oriented applications -- distributed systems. However, the amount of problems generated by physical distribution of systems, data bases and applications in coordination, control, data transfer, data redundancy, backup and recovery are so tremendous that they outweigh some of the predictable advantages which may be obtained through distributed systems. Despite the theoretical potential which exists for these systems, we are not observing a trend towards extensive implementation of distributed systems.

In this research, we propose to develop a general model for computer architectures which allows us to integrate some of the previous approaches for increasing computer capacity and which indicate new possibilities for improving the cost/performance ratio of future computer systems. The model leads to the development of general purpose data flow machines and to different degrees of sophistication within these machines.

The potential advantages for data flow machines has been recognized for over ten years. However, most of the research effort has been directed towards developing special purposes data flow machines (Rumbaugh [14], Dennis and Misunas [5,6], Gurd, et. al. [8], Schroeder and Meyer [16], Miller and Cocke [12]) and in developing new programming languages and representation methods which allow the reprogramming of applications for data flow machines (Miller [12], Miller and Rutledge [13], Cohen [2], Kosinski [10], Dennis [4]).

In this research, we propose a different approach. We have developed a hierarchical model which allows gradual change from the lowest level of the hierarchy, which is a single instruction stream sequential machine to the highest level, which is a multiple instruction stream with multiple processes and multiple transactions controlled through data flow processing. Since it is not necessary to reprogram existing applications, this leads to a more acceptable and structured approach.

The Underlying Assumptions

The assumptions used to develop the hierarchical model can be categorized as economic and technological. They are:

1) A steady increase in the demand for higher computation capacities which are needed to drive future computer communication systems and the application programs which are based upon them.

2) A demand for faster machines. This is due to the development of new types of applications based on Artificial Intelligence, Computer Assisted Diagnosis, Weather Forecasting, Oceanographic Research, Modeling of Economic Systems, and Decision Support Systems.

3) A high economic incentive exists for computer manufacturing companies to have a general purpose machine which can be used and marketed for a broad customer base. This is in constrast to such current machines as CRAY-I [15], STAR-100 [3] and ASC [17] which are special purpose and very limited in their customer base. Therefore, they are relatively expensive (see Russell [15] for data on the number of installed machines).

4) We expect a continuation in the present trend of decreasing cost for main memory, including associative memory. We expect to have future memories with a very high level of interleaving (almost complete concurrent access to different segments of memory), Wong and Tang [18].

5) A continued decrease in the cost of processors and logic components.

6) Due to the large accumulated investment in software and users' resistance to significantly alter their existing software and programming methods, computer architectures capable of supporting existing software while allowing a transition to more advanced languages are expected to be more acceptable.

7) Since we anticipate a continuing need for on-line systems, we expect an increased demand for high reliability and availability of computer systems.

The Basic Approach

The basic approach that we propose for future architectures is to decompose the CPU into two basic sets of elementary components - tracers and executors. Tracers detect the flow of instruction images in memory. This could be the program's instruction flow, as in present sequential machines, or tracing the data flow in future data flow machines. The second set is composed of executors which execute instructions provided to them by the tracers, and return results. Each executor is dedicated to one type of operation (e.g.,addition, multiplication, comparison, etc.) A computer may have multiple copies of the same executor, where the number of each type can be determined by the distribution of the demand for each instruction type (e.g., there may be 25 adders and 5 multipliers). The communication between tracers and executors is through a bus in which the images and data for instructions that are ready for execution are stored. The tracers fetch the instruction image and data for instructions that are ready for execution and place them on the bus. Each free executor examines the string of instructions which are on the bus. When an instruction image for this executor is detected, the instruction and data are removed from the bus and then executed.

Executors can be viewed as elements that alternate between idle periods and execution periods. In the idle period, the executor scans the bus for work to be performed. During the execution phase, it is disconnected from the bus until its operation phase is completed and then it renews the scanning of the bus.

The executors operate in parallel, independently from each other. The programs can share executors and be executed in parallel. The coordination between executors is accomplished through the memory (or the data). In this computer architecture, executors are passive elements. They are fed by the tracers, which are the only active elements.

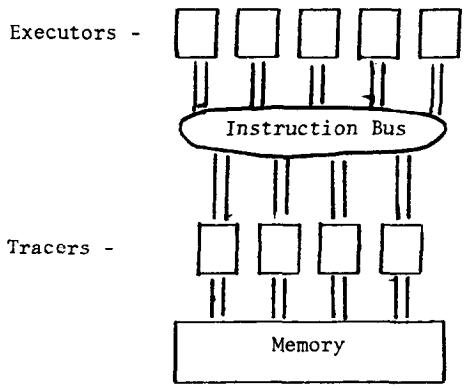

FIGURE 1

Relationship Between Executors and Tracers

The fact that executors operate independent of the bus allows us to have multiple executors for the same operation, where each operates independently. When an executor is busy executing and instruction for a given program, another executor can at the same time execute the same type of instruction for another program or a different portion of the same program. It is possible to have several parallel buses, each one dedicated to a subset of instruction types. This decreases the amount of data which is stored in each bus and decreases the path length from fetching the instruction and executing it.

Various Levels of the Computer Architecture Hierarchy

In this section, we demonstrate through several examples that by using the model introduced in the previous section it is possible to model existing and future computer architectures. These future computer architectures may be characterized by a hierarchy, which from preliminary analysis carry improved cost/performance ratios over existing machines. These architectures differ in the level of resource sharing and how tracers are assigned to programs, partitions or data items.

1) A Sequential Machine - Can be modeled as a machine containing one tracer and one executor for each instruction type. The tracer follows the instruction sequence and the next instruction is sent to the executor simultaneously. If a look-ahead feature is added, then more than one instruction can be executed but the initial ordering is preserved throughout the execution process.

2) A Multiprogrammed Machine - This machine has several tracers. Each is dedicated to a single partition of the memory and traces the instruction flow of the programs which are in its partition. An entire application program is stored in each memory partition.

There can be one or more executors per instruction type. The exact number depends on the statistical distribution and the usage frequency of the different instruction types. Some instructions will have a single executor while others may have multiple executors. The maximal level of multiprogramming is determined by the number of tracers. We make the assumption that simultaneous access can be made to the partitions but not within the partitions.

The performance of this machine will be the same order of magnitude as its level of multiprogramming, provided that no contention occurs at the executor level. The capacity of such a machine can be increased by adding tracers and memory to the basic configuration.

3) A Data Flow Machine (sequential within a data item) - This machine has many tracers and multiple executors for each instruction type. Each tracer follows the instructions that use and manipulate a specific data item (i.e., the tracer is assigned to a data item). Whenever the advancement of a tracer is blocked due to its waiting for another data item controlled by another tracer, it can be assigned to a different data item. As a result of this architecture, whenever the required data for an instruction image in memory is available, the instruction will be sent for execution independent of what happens in other processes.

The number of executors for each instruction type will depend on the statistical distribution of the demand for this instruction type, the environment and workload distribution at the user site. Sites will differ in the number of executors for each instruction type. For example, sites with a heavy demand for integer arithmetic will have more integer related exeuctors compared to sites which are heavily floating point oriented. Actually, it is possible for a user to begin with a minimal configuration that will contain few tracers and few executors. He can then monitor the demand (or workload) on the different components of his system and add executors for instruction types which are heavily loaded and are bottlenecks of the system. As his workload grows, he will be able to increase the capacity of his system by adding tracers, executors and memory.

In order to develop such an architecture, a different instruction format is necessary. Each instruction image and each instruction will contain:

a) Addresses of operands (actually only one is needed for the address of the data item associated with the first tracer that arrives).

b) Pointers to the next instruction that references this data item. Since a typical instruction references two data items as operands two pointers are necessary. The pointers will contain the addresses of the next instruction relative to the address of the present instruction, making the program relocatable in memory. For instructions containing a conditional branch, there will be a need for four pointers (two for each data item). The pointer will also contain an indication whether it is the first or second operand in the next instruction.

c) Two bits for synchronization purposes, each bit representing whether a data path has reached this synchronization point. When a tracer reaches an instruction, it checks the status of the other operand. If it is on, it will fetch the instruction, send it for execution and move to the next instruction on its chain. If it is off, it will set the bit that represents the availability of this data item from off to on and will move to the next instruction in this data chain. After the instruction has been executed, both bits are switched to off. Only if a tracer arrives at an instruction that modifies the content of its data item and the second data item for that instruction is not ready (i.e., the second bit is off) it will set the first bit on (to signal that the data item is ready for execution) and will be reallocated by the operating system to another task (data item). The order that tracers arrive at an instruction is irrelevant. The second tracer will always be the one that will activate the instruction (i.e., fetch the instruction image and data and send them for execution).

d) A bit representing whether the instruction is being executed. This is needed for synchronization purposes to prevent premature references to instructions due to loops or multiple references to the same instruction by multiple input data being processes in parallel.

The programming language for this machine could be the conventional programming languages that exist today. However, there will be a need to recompile the programs during which the data flows will be analyzed and the appropriate chains and pointers will be generated. In this machine, there will not be a need for branch instructions at the machine level since the instruction flow and sequencing is completely controlled by the data itself.

4) A Data Flow Machine (parallel within a data item) - This machine is similar to the data flow machine with sequential tracing on each data item except that all instructions are stored in an associative memory. During compile time, the program is analyzed and the instructions are detected in which the content of data items is changed. For each content change, a list of all instructions that reference this data item before its next change is prepared. During run time, when the content of a data item is changed, the appropriate list is retrieved and the proper bit in each instruction is set to on. A memory manager checks for all the instructions in which both data items bits are on (signalling an instruction ready for execution), retrieves them and sends them for execution. The difference between this architecture and the previous one is that here we parallel process instructions within the data item, thus reducing the elapsed time needed to execute a program.

5) Data Flow Machine with Multiple Copies of Items - In the previous two data flow machines, the control on the advancement of data items through their instructions was accomplished by using their synchronization lists. This prohibited the concurrent use of the same program by different sets of input data items (concurrent processing of multiple transactions by the same program, for example). This forces a sequential processing of transactions which belong to the same application program (even though the processing of the transaction itself is done in parallel). In applications where a high rate of transactions are generated per time unit, the fact that only one transaction can be processed concurrently by a program may be a bottleneck of the system, causing a long response time to transactions that use a specific program. The solution to this problem could be having multiple copies of the same program in memory. This is not desirable since it increases the memory required and the complexity of coordination and control between the different copies of that program.

Another solution is to use the data items for synchronization while the program will be used only for representing the instruction (operands) flow on those data item. The program will never change during its execution and thus multiple copies of the input data items will be able to use it.

An instruction image on this machine will contain:

a) The instruction code, representing the operation of this instruction. This is used to match the appropriate executor in the executor's pool.

b) Pointers to the next instruction that uses the data item (there will be two such pointers per instruction). Each pointer contains: (1) the relative address of the next instruction that references this data address); (2) an indication whether it is the first or second operand in the next instruction.

c) The difference between the addresses of the first and second data items which are used as operands in the instruction.

d) An extra location for each data item. Each location contains the address of the instruction that blocks the advancement of the process associated with this data item or it is empty.

During compile time, all data items which are changed during execution will be grouped together in a data block. The input data for the program will be part of the data block. A list of all data items that must be initially activated in that block in order to begin the execution process will be prepared during compile time and will be temporarily added to the data block. At the initiation of the execution process, a tracer will be assigned to each one of those data items and will use the instruction pointer chain to determine the next instruction that uses the data item. The next instruction will be fetched and the location of the second operand will be calculated (it will be in the data block). The instruction address associated with the second operand will be checked whether it is equal to the current instruction address. If it is, it will be fetched and the current instruction will be sent for

execution. If it is not, then the current instruction address will be entered to the data item which is the first operand and the tracer will be allocated another task. In that case, the first operand will be later activated by the tracer associated with the second operand.

The instructions associated with the program do not change during the execution time, and therefore multiple input data sets can use them concurrently. The purpose of the program in this case is to supply the operators and their ordering without controlling their execution, which is done in the data item level.

In this machine, the execution time of a program is increased in comparison to the execution time of data flow machines, which are instruction controlled, since at least one more access to memory for each instruction is needed. However, this may be worthwhile due to the fact that multiple transactions will be able to use the same program concurrently, and can therefore increase the throughput of those machines.

Purpose of the Research

Since this is an initial research effort to develop a model which is general enough to allow modeling of different computer architectures, we feel that many questions and possible research directions are open. One of the objectives of our research is to identify the important issues which are related to the model. Some of the immediate objectives are:

- To develop a more detailed and formal description of the hierarchical model for computer architectures.
- Classify current architectures under our model.
- Identify new architectures which will emerge as a result of the model.
- Identify problems in developing computer architectures that will emerge as a result of this research or previous research and analyze their impact on operating systems, programming languages, reliability, program termination and identification of control transfering.
- Estimate the costs and benefits of moving from one architecture level to another level.
- Identify the types of applications and environments which are best suited by each architecture level.

Expected Benefits from this Approach

The fundamental approach behind our research is to decompose the CPU into elementary components to allow a very high degree of resource sharing in the computer system. This leads to a model under which current and future computer architectures can be characterized. Some of the major benefits which may emerge as a result of our research are:

- If our previous observations are correct, there will be a high motivation for the computer industry to invest in these types of machines, which may result in significant advantages to the computer users community.
- Since the system is composed of relatively simple special purpose components (tracers and executors), it is expected to have improved cost/performance over current sequential or parallel processing machines.
- It is expected that our model can be used for comparing different architectures and for analyzing their differences.
- The proposed architectures tailor themselves to the applications and to their environment, versus existing architectures, in which the applications have to be tailored to the hardware. Once the basic system is installed, it will be possible to significantly increase the capacity of the basic system and to reduce bottlenecks in the system with little extra cost.
- The model allows the possibility of a gradual change from the existing sequential machines to a data flow machine, thus making the data flow machine more attractive.
- The high level of parallelism, duplication and self operation of each one of the basic components points to the possibility of developing self-checking procedures. Several components can be used to check other components of the machines and can activate appropriate automatic self-correction procedures in order to take care of malfunctioning by shutting off damaged executors or tracers. The basic question, of course, is what type of procedures should be developed in order to identify the malfunctioning components.
- Our architecture supports existing programming languages; there is no necessity for reprogramming. It is true that to attain the full advantage of these machines reprogramming will be necessary. However, even without reprogramming, we would not be surprised if one order of magnitude of improvement in the cost/performance will be obtained.

References

1. Barnes, G.H., R.M. Brown, M. Kato, D.J. Kuck, D.L. Slotnick and R.A. Stokes, "The ILLIAC IV Computer," IEEE Trans. Computers C-17, No. 8 (August 1968), pp. 746-757.
2. Cohen, Ellis S., "Semantic Models for Parallel Systems," Department of Computer Science, Carnegie-Mellon University, January 1975.

3. Control Data Corporation, Control Data STAR-100 Computer Hardware Reference Manual, 1974.

4. Dennis, Jack B., "First Version of a Data Flow Procedure Language," Massachusetts Institute of Technology Project MAC, Computation Structures Group Memo 93-1, August 1974.

5. Dennis, Jack B. and David P. Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," Massachusetts Institute of Technology Project MAC Computation Structures Group Memo 102, August 1974.

6. Dennis, Jack B., David P. Misunas and P.S. Thiagarajan, "Data Flow Computer Architectures," Massachusetts Institute of Technology Project MAC, Computation Structures Group Memo 104, August 1974.

7. Enslow, Philip H. Jr., "Multiprocessor Organization - A Survey," Computing Surveys, Vol. 9, No. 1, March 1977.

8. Gurd, J.R., P.C. Treleoren and I. Watson, "A Data Flow Computer Architecture," Working Paper, University of Manchester, 1977.

9. Keyes, Robert W., "Physical Limits in Digital Electronics," Thomas J. Watson Research Center Yorktown Heights, New York, RC5169, December 1974.

10. Kosinski, Paul R., "A Data Flow Language for Operating Systems Programming," Proceedings of ACM SIGPLAN-SIGOPS Interface Meetings, SIGPLAN Notices 8 and 9, September 1973.

11. Miller, Raymond E., "A Comparison of Some Theoretical Models of Parallel Computation," IEEE Transactions on Computers, Vol. C-22, No. 8, August 1973.

12. Miller, Raymond E. and J. Cocke, "Configurable Computers: A New Class of General Purpose Machines," Lecture Notes in Computer Science, Vol. 5, Springer-Verlag, 1974.

13. Miller, Raymond E. and J.D. Rutledge, "Generating a Data Flow Model of a Program," IBM Tech. Disclosure Bulletin, Vol. 8, 1966. pp. 1150-1153.

14. Rumbaugh, James, "A Data Flow Multiprocessor", Proceedings of 1975 Sagamore Computer Conference on Parallel Processing.

15. Russell, R.M., "The CRAY-1 Computer System," CACM, Vol. 21, No. 1, January 1978, pp. 63-72.

16. Schroeder, M.A. and R.A. Meyer, "A Distributed Computer System Using a Data Flow Approach," Proceedings of the 1977 International Conference on Parallel Processing.

17. Texas Instruments, Inc., A Description of the Advanced Scientific Computer System, Austin, Texas, April 1973.

18. Wong, C.K. and D.T. Tang, "Dynamic Memories with Faster Random and Sequential Access," IBM Research, Computer Sciences Department, York-Town Heights, New York, RC5682, October 16, 1975.