An architecture for doing concurrent systems research

by VASON P. SRINI
University of California
Berkeley, California

ABSTRACT

An architecture for experimenting with different algorithms in numeric/symbolic computing, concurrent programming paradigms, and interconnection networks is proposed. The architecture comprises a large collection of processors organized in groups called levels. Each level has interconnection networks to communicate with memory units in the level and also a global memory called system memory. Each memory unit in the architecture has two parts, data memory and synchronizing/status memory. The latter is identical to the former except for the word-size. Separating synchronizing/status information from data reduces memory contention and improves the cycle time to memory for reading or writing.

Two proposals for processing elements are also outlined. One of them is a custom design and the other is a commercial processor with the addition of functional units capable of doing floating-point and symbolic calculations. The architecture will have facilities to support experimentation with three classes of concurrent computation models, a task level dataflow, demand driven, and cooperating sequential processing; programming languages such as Prolog with AND parallelism and built-ins for scientific computation will be supported by the architecture along with other languages. An initial design of the architecture using crossbars as interconnection networks is also outlined.
INTRODUCTION

Concurrency is a key area in computer science that can use VLSI technology to achieve improved performance. Concurrency can be specified in the model of computation and algorithms, and supported by the computer architecture. Although concurrency has been studied in detail, there are very few facilities for experimenting with concurrent architectures, interconnection structures, and software. Existing systems such as Cm* at Carnegie-Mellon University and TRAC at the Univ. of Texas in Austin are difficult to program due to programs and data distribution problems. For example, the Cm* architecture supports fast, slow, and very slow memory. Distributing programs and data so that time to obtain data from memory can be reduced is difficult. In the TRAC architecture, the memory cycle time is slow because of the switch between memory and processors. Furthermore, both lack the potential to evaluate the execution overheads of different models of computation. This paper describes a computer systems research facility that can support concurrent systems research in scientific computing aided by symbolic/logic processing. The objectives of the proposed computer system are:

1. To develop a system using several hundred processors to support the execution of numeric and symbolic algorithms coded in functional, logic, and control flow programming styles
2. To evaluate interconnection structures such as integrated bus-star, integrated bus-ring, integrated ring-star, global bus, crossbar, omega network, and banyan
3. To develop strategies for distributing programs and data in a semi-automatic way
4. To determine the overhead in executing programs on concurrent architectures and design tradeoffs related to this overhead
5. To develop concurrent programs to solve problems in areas such as fluid dynamics, aerodynamics, weather prediction, computational physics (e.g., simulating the superconductive super collider (SSC)), and computational chemistry, and to develop CAD tools for designing concurrent architectures

An outline of the systems architecture and the computation models that will be supported by it are discussed next. Two key features of the system are the separation of synchronizing/status memory from data memory, and the use of clusters of processors to support spatial and temporal locality in programs.

SYSTEMS ARCHITECTURE

There are three major models of computation that have the characteristics of concurrence and asynchrony; they are dataflow, demand driven, and Hoare's cooperating sequential processes. The dataflow and demand driven models also have functionality; that is, the execution of an activity represented as a node in the dataflow graph consumes inputs and produces outputs. There are no side-effects in the execution. Hence, program maintenance and distribution can be done without excessive overhead. An organization of processors that is expandable in a modular way and capable of supporting the three models of computation is shown in Figure 1. The synchronizing/status memory parts of memory units are not shown in the figure. The shadow of the drawing (obtained by duplicating the entire figure) can be thought of as the synchronizing/status part. Implementation is planned for the support mechanisms needed for the execution of programs on the processors in Figure 1 as the kernel functions of an operating system. The support functions accomplish task allocation, token communication, and node reassignment when processors fail. There are several reasons for using software instead of hardware to implement the support functions. A key reason is that experiments can be conducted using different policies for the support mechanisms. The kernel functions will be executed on a subset of (three or four) processors in each level. The kernel functions needed to execute programs using the dataflow model of computation are shown in Figure 2. By changing the semantics of the kernel functions, different strategies for task allocation, token construction, and token communication can be studied. The overhead in executing tasks, and the design tradeoffs related to the overhead can also be studied. By changing the structure of the kernel, the other models of computation can be supported. A detailed description of the organization of processors in Figure 1 and the kernel functions in Figure 2 are in the papers by Srini.

The organization of processors into levels allows high speed interconnection networks such as crossbars to be used for IN1-IN4 in Figure 1. It is believed that 32 x 64 crossbar chips with 400 pins can be fabricated and packaged using state-of-the-art technology. The levels of processors also support locality and simultaneity of execution in concurrent programs. For example, blocks of code operating on data structures such as arrays and lists contain spatial locality. Unfolding of loops and having many instances of the loop in concurrent execution provide temporal locality.

There are several programming paradigms that can be realized using the above system. We will focus on functional pro-
Figure 1—Organization of processors

PD - Peripheral Devices
C - Compiler
IN - Interconnection network
UI - User Interface
P - Processor
gramming, logic programming, and control flow programming because they have the potential to express concurrency. The mapping from the programming paradigms to the three models of computation are explained by Vegdahl and Treleaven. The purpose of the proposed architecture is to support the experimentation of multiple computation models with different programming paradigms. For example, functional programs will be executed using dataflow, demand-driven, and control flow architectures, and the execution times will be analyzed to determine the overheads and bottlenecks due to the computation model. Similarly, logic programs with AND, OR, and stream parallelism can be executed using dataflow and control flow architectures, and the execution times can be analyzed to determine the overheads. As a result of the experiments, high performance architectures suitable for the three programming paradigms will emerge.

**ORGANIZATION OF PROCESSORS**

There are several possible ways to implement the logical organization of processors shown in Figure 1. One implementation was suggested in the dataflow processor proposal. Another implementation is currently being investigated by researchers headed by Despain at the University of California at Berkeley. The key ideas are the use of VLSI chips for the interconnection network to reduce delay and wiring complexity, the use of processors with specialized functional units to support the computation models, and the use of operating system functions to perform the flow control. Memory is separated into data part and synchronizing/status part to reduce memory interference and speed up synchronization. The word length for the data part is 32 bits, and for each word there is a byte in the synchronizing/status part. The memory units in the proposed system are byte addressed.

An initial design for the architecture in Figure 1 is outlined below. The numbers chosen were based upon the feasibility of constructing the architecture and its usefulness for experimentation. The architecture uses crossbars instead of buses and rings to get fast access to all the memory units so that communication costs do not clutter other overhead issues. With a crossbar, it is possible to get the interconnection delay to be around 20% of the memory cycle time. The memory contention problem in the crossbar will be handled in a distributed way at the processors. A processor experiencing memory contention will proceed with other activities and re-attempt at a later time. It is believed that a crossbar with 50 nanoseconds delay can be fabricated by a corporation such as NCR using an advanced CMOS process, and that 256K memory chips with a cycle time of 250 nanoseconds will be cheap in 1985. The number of processors and memory units in a level are dictated by the size of the crossbar. The initial design of an 8 × 8 crossbar chip leads us to believe that a 16 × 32 crossbar chip can be designed and fabricated in 1985. The size of the memory units are determined by the memory requirements of application programs in the scientific computations. All the data paths are assumed to be 32 bits wide. This is not a restriction for doing some of the scientific calculations involving 64-bit data items. The data paths can be readily expanded by adding 32 single-bit VLSI crossbar chips to each interconnection network in the architecture.

1. Each level memory is a set of 32 memory modules, each with a 4 Mbyte capacity and two ports. The second port can be used for moving data to and from I/O devices. Corresponding to each of the above modules, there is a module in the synchronizing/status memory with a capacity of 1 Mbyte (one byte for each 32-bit word in the data memory). The 4 Mbytes of data memory can fit on the single hex board used by DEC if 256K memory chips are used. These boards are used in the VAX machines and are commercially available.
2. Each level has 16 processors; most are general purpose and some are special purpose processors.
3. The system memory is a shared one, having 12 ports (eight ports for eight levels of processors and four for I/O processors and front-end) for each module. It contains up to 64 modules, each with a capacity of 16 Mbytes. Corresponding to each of these memory modules is a memory module in the synchronizing/status memory with a capacity of 4 Mbytes and 12 ports. The 16 Mbytes of data memory can fit on a single DEC hex board if 1M memory chips are used. These boards are expected to be commercially available in 1986. The 12 ports can connect up to eight levels of processors, three input and output processors (see PD in Figure 1), and one interface to the front-end running the compiler. The system memory has four 16 × 64 crossbars to connect the I/O and front-end processors with the data part of the system memory. An additional set of four 16 × 64 crossbars connect the above processors to the synchronizing/status memory part of the system memory. This organization allows high speed communication of data to I/O devices.
4. The interconnection network IN1 contains a 16 × 32 crossbar with 32 bits wide data and address paths to the data part of level memory. It also contains a 16 × 32 crossbar with an 8-bit-wide data path and a 32-bit-wide address path to the synchronizing/status part of level memory.
memory. The crossbar will be built using single-bit-slice chips. Each chip contains 182 pins. This pin count is based on the Macpitts design of the chip. Macpitts is a silicon compiler developed by the Lincoln Laboratory and modified by researchers at the University of California at Berkeley to generate layout for special purpose applications. An interconnection network similar to IN1 can be used on the second ports of the level memory modules to connect the I/O controllers and devices.

5. The interconnection network IN2 is a 16 x 64 crossbar with 32-bits-wide data and address paths to the data part of system memory. It also contains a 16 x 64 crossbar with an 8-bit-wide data path and 32-bit-wide address path to the synchronizing/status part of system memory. This crossbar will also use single-bit-slice chips. Each chip needs 263 pins. This pin count is also based on the Macpitts design of the chip.

6. The interconnection network IN3 contains a 16 x 32 crossbar with 32-bit-wide data and address paths and a collection of 16 console debuggers, one for each processor in a level. The 32 output ports of the crossbar will be used in the following way. Sixteen ports will be connected to the processors and 16 to console debuggers. Thus, the crossbar in IN3 allows processors to communicate within a level. Using the console debugger, a processor can communicate with the external environment. There is another 16 x 32 crossbar to connect the synchronizing/status ports of the processors (see LB paths in Figure 3), and console debuggers.

The efficiency of communication facilities is an important area in concurrent systems research. For example, there are three interconnection networks in Figure 1. The performance of the computer system shown in Figure 1 depends on the interconnection networks, processor architecture, and the kernel functions. By using different interconnection networks for IN1-IN3, and separating data from synchronization/status information, the efficiency of communication facilities can be determined. It is believed that the above implementation using crossbar will provide insight into predicting the efficiency of other communication facilities such as the Omega network. The architecture will also allow us to study the performance gain obtained by separating the data and synchronization/status information and keeping them in separate memory units.

PROCESSOR DESIGN

Efficient execution of programs containing numeric and symbolic computation requires processors with an instruction set containing logical, fixed-point, floating-point, string manipulation, and sorting operations. Other features required are data cache, mechanisms for communicating with memory, and fault diagnosis. There are at least two ways to develop processors with the above characteristics; one is to custom design a processor, the second is to use a commercially available processor and interface special purpose units to it.
**Custom Design**

The block diagram of a custom designed processor that satisfies the above requirements and supports the dataflow model is shown in Figure 3. The processor is capable of executing dataflow graphs generated from functional programming languages and logic programming languages. The HB paths in the processor communicate data. The LB paths communicate synchronizing/status information. By employing user microprogrammable functional units in the processor, it will be possible to experiment with different instruction sets for the processor. Some of the key features of the processor and the units are explained next. For further details, see my previous description of this processor.

The processor receives from a memory unit an enabled node for firing as a sequence of packets forming a variable length message (token). Data may be contained in the token, or it may arrive separately in another token. A node's operation can contain a maximum of five instructions. The processor performs node firing and sends the results to other processors as specified by the kernel functions doing the allocation of nodes to processors. The various parts of the message-based processor shown in Figure 3 are now described. All units of the processor will eventually have built-in diagnostics and perform self-diagnosis. Diagnostic information will be communicated to the diagnostic unit.

**Message (MSG) unit:** This unit receives and sends tokens containing data (synchronizing/status information) to the interconnection network, and stores it in the buffer/ports unit.

**Decoder:** This unit decodes instructions received in a token, and sends control to the router for supplying operations and data to the various functional units.

**Router:** The operations and data supplied by the decoder are routed to the functional units using a receive/ack/last protocol. Three control lines are used for the protocol to each of the seven functional units.

**Diagnostic unit:** This unit communicates with all the parts of a processor to gather diagnostic information. It decides whether the processor is healthy or faulty on the basis of the gathered information. Urgent tokens containing diagnostic information from memory are received by this unit. Diagnostic information is sent to memory and to other processors in the form of urgent tokens. This unit also reroutes tokens stored in the buffer/ports unit. Note that the diagnostic unit is not the hard core of a processor, since the functional units do their own diagnostics.

A bus architecture is used in the processor for communication between the table memory, floating-point, fixed-point, logical, and string units. The functional units are user microprogrammable so that new instructions can be added and microdiagnostics can be used. The processor has been designed using AMD 2900 bit-slice chips to get estimates on the execution times for different instructions. A VLSI version of the processor will be custom designed employing Weitek's 1064/65/66 floating-point chips.

**NCR/32—PLM Design**

The block diagram of a processor using a commercially available microprocessor is shown in Figure 4. It contains an...
The built-in functions of Prolog that are related to scientific unification of a goal's arguments with a clause-head's the context for backtracking, discard the current environment for functional and control flow programming languages can be obtained by starting with Prolog. A Prolog program execution unit called PLM \( ^{18} \) is planned to support symbolic processing. An extended arithmetic unit (EAU) is included to perform integer multiply and divide, floating-point operations, and trigonometric functions. A version of this processor is under development at the Computer Science Division of the University of California at Berkeley. \(^{18}\)

The PLM executes the instructions of an abstract machine developed by Warren. \(^{19}\) The PLM has five classes of instructions to execute Prolog programs and to invoke the NCR/32 to do scientific computations. Some of the instructions perform unification of a goal's arguments with a clause-head's arguments, moving of a goal's arguments in registers to memory or other registers, and moving of the arguments of a goal in a clause's body from memory to registers. Other instructions create a new environment for a procedure call, save the context for backtracking, discard the current environment, backtrack, and branch instructions based on the data type in an argument register to select a clause in a procedure. The built-in functions of Prolog that are related to scientific computation are not performed by the PLM. Rather, an escape mechanism is provided in the PLM architecture to communicate the built-ins to the NCR/32. The built-in functions are computed using the NCR/32 and the EAU. A compiler written by Van Roy \(^{30}\) generates code for the PLM from Prolog programs.

The NCR/32-based design is an inexpensive design created to aid in understanding some of the research issues in interconnects processors to memory and managing an efficient software. To provide high performance in doing scientific computations, the convex C-1 processor \(^{21,22}\) can be used. It is a 64-bit processor with a virtual memory space of 2 Gbytes per process and a maximum of 2 Gbytes of physical memory. One half of the physical memory is reserved for I/O. The CPU and I/O communicate memory using two 80 Mbyte buses (64-bit data path) and a memory controller. It appears that an interconnection network can be placed between the CPU and memory as well as between the I/O and memory without degrading memory accesses. The synchronizing/status memory can be added by tapping the addresses that appear on the two buses and sending them to the synchronizing/status memory, or by using the I/O space. Because Unix 4.2 bsd is running on the Convex C-1 processor, most of the software developed for the NCR/32-based design can be used.

**Support For AND Parallelism**

Concurrency in Prolog programs can be exploited by using AND, OR, and stream parallelism. \(^{23}\) Exploiting OR and stream parallelism in Prolog programs needs hardware support because supporting them in software requires excessive overhead. It is also difficult to control OR parallelism. Initially, AND parallelism in Prolog programs will be considered. At a later stage, stream parallelism will be exploited. The goals in the body of a clause with AND parallelism will be concurrently executed on a set of processors in a level.

Compiling Prolog programs with AND parallelism to the architecture presents some additional problems; the first is detecting the AND parallelism at compile time, the second is generating code for multiple processors that would keep track of the environment and the choice points without creating excessive memory interference. Although dynamic analysis of Prolog programs to detect AND parallelism has been reported by Conery, \(^{23}\) it is an expensive approach and complicates the code generation for multiple processors. Recently, Chang \(^{24,25}\) has devised a scheme to detect AND parallelism at compile time by annotating the programs at the top level. The above development has opened new possibilities in code generation. Figure 5 shows a sample Prolog program and the dataflow graph that can be generated for it. The data dependencies between the goals in a body are shown by directed arcs. These arcs resulted from the compile time analysis of the Prolog program. The dashed arcs in the graph represent communication paths for synchronization/status information. These arcs can also be supplied by the compiler but they require extra analysis. The nodes in the dataflow graph can be assigned to processors in a single level or multiple levels for node firing. By using a tagging scheme and special operators.
An Architecture for Doing Concurrent Systems Research

\[
g(X,Y) :- p(X,Z), p(Z,Y).
\]

QUERY:

\[
p(john, dick).
\]

\[
p(john, nancy).
\]

\[
p(dick, robert).
\]

\[
p(john, Q).
\]

Figure 5—(a), Prolog program; (b), dataflow graph for a Prolog program

Application Areas

Programs from three application areas will be executed on the architecture to evaluate the performance of the system. The areas are scientific, knowledge-based expert systems, and CAD tools (e.g., silicon compilers, circuit simulators, routers, and placement programs for cells). The programs will be written in extended Prolog and other languages.

CROSSBAR DESIGN

One of the components that play a key role in the performance of the architecture in Figure 1 is the crossbar. Although crossbars have been used in computer systems for more than two decades, very few high-performance off-the-shelf chips are available. Some VLSI designs based on an incremental design have been proposed by Franklin \cite{26,27} and other designs are internal to corporations making signal processing systems. In this paper, a high-performance fixed-delay (assuming conflict-free references) design for \(8 \times 16, 16 \times 32, 16 \times 64,\) and \(32 \times 64\) crossbars is proposed. The design employs a single-bit-slice approach with a maximum of six gate delays in the switch part (not including delays in the pads and pad-drivers) and a maximum of 400 pins.

The design and the functioning of the crossbar is explained using the pin out of a single-bit-slice crossbar chip with 16 input ports and 32 output ports, as shown in Figure 6. The input ports are connected to processors, and the output ports can be connected to memory units or processors. For example, if the processors correspond to those shown in Figure 3, then the input ports will be connected to the HB-senders or LB-senders and the output ports will be connected to the HB-receivers or LB-receivers. Each input port contains five address pins, one address/data-in pin, and one data-out pin. Each output port contains a data-in pin and a data-out pin. Processors can communicate words of data and address by using multiple single-bit-slice chips. For example, by stacking 32 single-bit-slice chips, a 32-bit address/data can be sent to memory and a 32-bit data received from memory.

The design makes several simplifying assumptions. If two or more processors request the same memory unit, a simple policy is used to select one of them. One or more other processor(s) will receive signals indicating contention for memory. The processors are assumed to have the necessary logic such as "A + SWITCH" to save an old context, create a new context, and branch, recursive clauses and backtracking can be handled. Unification (a process of finding the most general common instance of two terms) is carried out by using nodes with the operation "UNIFY". Creating an environment for evaluating a literal (sub-goal) is done by using a node with the operation "BIND". Microroutines have to be developed for the above operations and others that support AND parallelism.

Figure 6—Pin out for a single-bit-slice crossbar chip
to handle contention and retry at a later time. Each processor is assumed to supply the memory unit or the processor number it wants to communicate, followed by an address and data in the case of a memory write, or just an address in the case of a memory read. The single-bit-slice chip in Figure 6 contains several multiplexers, selectors, and arbitration logic designed in static CMOS. The NAND gates used in the design have a fanin of 5 and the NOR gates have a fanin of 12.

A processor wishing to read/write from one of the 32 memory units supplies the address of the memory unit on the address pins. The processor then sends the address of the desired location in the memory unit using the address/data-in pins of the stack of chips. For write operations, the processor supplies data on the next processor cycle using the address/data-in pins. For read operations, the processor receives data from the selected memory unit on the data-out pins of the stack of chips. Each memory unit is connected to one output port (a data-in pin and a data out-pin) of a single-bit-slice chip. A memory unit receives an address followed by data from the data-out pins of the chips if a write to memory is to be performed. If an address is not followed by data, then the memory unit reads the contents of the desired location and supplies it on the data-in pins of the chips.

Note that the proposed design is different from the incremental design in two respects. The flexibility of the incremental design is traded off for reduced gate delay. The complexity involved in contention handling is traded off to obtain a simple design. The price one has to pay for simplicity is degradation in performance when processors requests are skewed towards a subset of memory units. Assigning tasks to processors so that memory references will not be skewed towards some memory units is still an open problem. It appears that reassigning tasks to other processors at runtime can mitigate the problem.

DISCUSSION

An architecture capable of supporting 128 processors is proposed. The architecture employs 16 x 32 crossbars and 16 x 64 crossbars for processor memory communication. VLSI chips for the crossbars seem feasible with static CMOS technology (2-micron feature size and two layers of metal) and pin grid array packaging. The use of crossbars in the system allows the cycle time to read (write) from either level or system memory to be the same. It is expected that having one cycle to memory should reduce the execution overhead and simplify program and data distribution. This feature of the proposed system should be contrasted with the fast, slow, and very slow memory accesses in CMOS technology which makes program distribution a challenging task.

The proposed architecture can be used to map many of the multiprocessor architectures proposed by universities and industrial laboratories. For example, the Cedar multiprocessor system of the University of Illinois can be mapped on to the architecture if each cluster is treated as a level of processors and the number of processors per cluster is limited to 16. The Omega networks will be replaced by the crossbars. The cluster control unit and the global control unit functions can be implemented using the synchronization/status memory units and one processor in each cluster. The mapping of Cedar architecture is important because thousands of benchmark programs analyzed by the Paraphrase compiler can be run on the architecture. The results will allow us to determine the performance penalty paid for multistage blocking networks such as Omega. The execution overheads for task level dataflow on control flow architecture and dataflow architecture can be compared. The static dataflow architecture of Dennis can also be implemented on the architecture if the enabling count information of nodes are kept in the synchronizing/status memory units, and a few of the processors in each cluster execute functions corresponding to enabling condition detection.

Several issues must be resolved before constructing the proposed system. A key issue is the protocol to be used in reading or writing to data memory with the synchronizing/status memory in place. Another key issue is predicting the traffic between the processors and the memory for a class of algorithms and developing appropriate policies to be used in distributing programs and data. Another issue is the accessing of data structures (e.g., arrays, lists, and records). Efficient ways to read and update data structures must be devised. This is still a challenging problem in the dataflow and demand driven architectures. New languages have to be developed or existing ones augmented to exploit the multiple processors in the architecture. Another key issue is the design of the operating system, and predicting the overhead in the execution of programs on the processors.

Developing a detailed event-driven simulator for the architecture will be the first step of the project. The availability of VAX-11/784 with shared memory at LBL will allow us to get a head-start on the simulation activity. An analyzer for detecting AND parallelism in Prolog programs and a compiler for generating dataflow graphs are two other areas that will be carried out on the VAX-11/784.

ACKNOWLEDGMENTS

I am thankful to Professor Al Despain for his comments on the architecture and the crossbar design. The numerous discussions I had with Tep Dobry, Barry Fagin, Wayne Citrin, Jung Chang, Philip Bitar, and Robert Young have been invaluable in my understanding of Prolog and the PLM design. I am also thankful to NCR Corporation, Dayton, for funding parts of the Prolog machine research. I am thankful to Dr. Arie Shoshani for his comments on the paper and the discussions related to the architecture.

REFERENCES


