CUCS-43-83

ARCHITECTURE AND APPLICATIONS OF DADO: A LARGE-SCALE PARALLEL COMPUTER FOR ARTIFICIAL INTELLIGENCE

SALVATORE J. STOLFO
DANIEL P. MIRANKER
DAVID ELLIOT SHAW

Architecture and Applications of *DADO*: A Large-Scale Parallel Computer for Artificial Intelligence*

Salvatore J. Stolfo
Daniel Miranker
David Elliot Shaw

Columbia University

Abstract

As part of our research on very high performance parallel architectures, we have been investigating machine architectures specially adapted to the highly efficient implementation of artificial intelligence (AI) software. In the course of our research we designed *DADO*, a highly parallel, VLSI-based, tree-structured machine, and implemented a high-speed algorithm for production systems on a simulator for *DADO*. Subsequent research has convinced us that *DADO* can support many other AI applications, including the very rapid execution of *PROLOG* programs, and a large share of the symbolic processing typical of contemporary knowledge-based systems. In this brief report, we outline the hardware design of a moderate size *DADO* prototype, comprising 1023 processing elements, which is currently under construction at Columbia University. We then sketch the software base being implemented on a small 15 processing element prototype system including several applications written in *PPL/M*, a high-level language designed for specifying parallel computations on *DADO*.

1. Introduction

As knowledge-based systems begin to grow in size and scope, they will begin to push conventional computing systems to their limits of operation. Even for experimental systems, many researchers reportedly experience frustration based on the length of time required for their operation. For applications requiring real-time response from an expert system (for example, electronic warfare) conventional implementations may not be practical. Recently, several AI researchers (see [Nilsson 1980], for example) have suggested that significant increases in the performance of contemporary AI systems might be realized through distributed processing or the use of specialized parallel hardware. Some attention has been given to issues of parallelism in system organizations for cooperating distributed AI subsystems [Lesser and Corkill 1979]; special hardware for high speed property inheritance and related operations in systems based on semantic network-like formalisms [Fahlman 1979, Hillis 1982]; and the design of machines supporting the parallel execution of certain relational algebraic operations having practical importance in large-scale knowledge-based systems [Shaw et al. 1981]. The potential applications of *very large scale hardware parallelism* to the execution of AI systems, however, has remained largely unexplored.

In this paper, we describe *DADO* [Stolfo and Shaw 1982], a tree-structured, multi-processor based architecture that utilizes the emerging technology of VLSI systems in support of the highly efficient parallel execution of large-scale AI systems. In [Stolfo and Shaw 1982], we reported a high-speed algorithm for *Production System* programs, which has been implemented on a simulator for *DADO*. Production Systems form the basis for a wide range of approaches to building knowledge-based expert systems. Subsequent research has convinced us that *DADO* may support many other AI applications including the very rapid execution of *PROLOG* programs and a large share of the symbolic processing typical of knowledge-based systems. A small (15 processor) prototype of the machine, constructed at Columbia University from components supplied by Intel Corporation, is operational. Based on our experiences with constructing this small prototype, we believe a larger prototype, *DADO2*, comprising 1023 processors and capable of significant performance improvements over implementations based on von Neumann machines, to be technically and economically feasible for implementation using current technology. We believe that this experimental device will provide us with the vehicle for evaluating the performance, as well as the hardware design, of a full-scale version of *DADO* implemented entirely with custom VLSI circuits.

2. The *DADO* Machine Architecture

A full-scale production version of the *DADO* machine would comprise a very large set (on the order of hundreds of thousands) of *processing elements* (PE's),

*This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-82-C-0427, as well as Intel Corporation, Digital Equipment Corporation, IBM Corporation and Valid Logic Systems. We gratefully acknowledge their support.

each containing its own processor, a small amount (2K bytes, in the current design of the full-scale version) of local random access memory (RAM), and a specialized I/O switch, which will be implemented using a custom integrated circuit. The PE's are interconnected to form a *complete binary tree*.

Within the *DADO* machine, each PE is capable of executing in either of two modes. In the first, which we will call *SIMD mode* (for *s*ingle *i*nstruction stream, *m*ultiple *d*ata stream), the PE executes instructions broadcast by some ancestor PE within the tree. In the second, which will be referred to as *MIMD* mode (for *m*ultiple *i*nstruction stream, *m*ultiple *d*ata stream), each PE executes instructions stored in its own local RAM, independently of the other PE's. A single *control processor* (CP), adjacent to the root of the *DADO* tree, controls the operation of the entire ensemble of PE's.

When a *DADO* PE enters MIMD mode, its logical state is changed in such a way as to effectively "disconnect" it and its descendants from all higher-level PE's in the tree. In particular, a PE in MIMD mode does not receive any instructions that might be placed on the tree-structured communication bus by one of its ancestors. Such a PE may, however, broadcast instructions to be executed by its own descendants, providing all of these descendants have themselves been switched to SIMD mode. The *DADO* machine can thus be configured in such a way that an arbitrary internal node in the tree acts as the root of a tree-structured SIMD device in which all PE's execute a single instruction (on different data) at a given point in time. (A PE in SIMD mode may also be instructed to *disable* itself, in which case instructions placed on the broadcast bus will be ignored by the PE, while descendants of the PE who remain *enabled* will receive and execute such instructions.) Thus, *DADO* provides direct support for large-scale MSIMD (multiple SIMD) execution. This flexible architectural design supports the logical division of the machine into distinct partitions, each executing a distinct task.

The *DADO* I/O switch, which will be implemented in custom VLSI and incorporated within the 1023 processing element version of the machine, has been designed to support communication between physically adjacent tree neighbors, as well as communication between PE's that are adjacent in a logical linear ordering embedded within the tree. In addition, a specialized combinatorial circuit incorporated within the I/O switch will allow for the very rapid selection of a single distinguished PE from a set of candidate PE's in the tree. Currently, the 15 processing element version of *DADO* performs these operations with the sequential logic embodied in its off-the-shelf components.

In the following sections we outline the hardware design of the *DADO* prototypes and then describe *PPL/M*, a variant of the *PL/M* language, providing several primitives for specifying parallel computation on *DADO*. In the concluding sections we overview the applications currently under active investigation for implementation on *DADO*.

3. The *DADO* Prototypes

A 15-element *DADO* prototype, constructed from (partially) donated parts supplied by Intel Corporation, is currently operational. The 15 PE's are integrated on two wire-wrap prototype boards supplied by Valid Logic Systems. A much larger version, *DADO2*, is under construction which will incorporate 1023 PE's constructed from two commercially available Intel chips. The *DADO2* prototype consists of 32 printed circuit boards, each containing 32 PE's, housed in a rack which can fit in either a DEC VAX 11/750 cabinet, or an IBM Series I cabinet.

3.1. The Prototype Processing Element

Each PE in the prototype systems incorporate an Intel 8751 microcomputer chip, serving as the processor and an 8K X 8 Intel 2186 RAM chip, serving as the local memory. (Two simple logic gates are used to properly integrate the RAM and processor.) Although the full-scale version of *DADO* has been designed to incorporate a 2K RAM within each PE, an 8K RAM was chosen for the prototype PE to allow a modest degree of flexibility in designing and implementing the software base for the full version of the machine.

The Intel 8751 is a powerful 8-bit microcomputer incorporating a 4K eraseable programmable read only memory (EPROM), and a 256-byte RAM on a single silicon chip. One of the key features of the 8751 processor is its I/O capability. The four parallel, 8-bit ports provided in a 40 pin package has contributed substantially to the ease of implementing a binary tree interconnection between processors.

In the 15-element prototype design, the communication primitives and execution modes of a *DADO* PE are implemented by a small *kernel system* resident within each processor EPROM. The specialized I/O switch envisioned for *DADO2* is simulated in the smaller version by a short sequential computation. Based on our estimates of minimum clock speed, *DADO2* will be capable of executing in excess of 600 million instructions per second, assuming inter-processor communication to be implemented with a combinational logic I/O switch. Although pipelined communication is employed in the kernel design, it is expected that only 150 million instructions per second would be achieved using the current design. Thus, the design and implementation of a

custom I/O chip forms a major part of our current research activities.

3.2. The PE kernel

As noted, the 4K EPROM of the Intel 8751 stores the system kernel of a PE, which includes code performing the most basic communication and synchronization functions as well as the simulation of SIMD and MIMD modes of execution. The kernel system is designed in such a way as to logically divide the 8K RAM space of the Intel 2186 chip into two portions for each of the execution modes. The initial 1K portion, referred to as *SIMD RAM*, is a reserved data space for variables and constants operated upon by a PE while in SIMD mode. The remaining 7K portion of RAM is used for storage of code, as well as the local variables used during the MIMD mode of operation.

4. Programming DADO

PL/M [Intel 1982] is a high-level language designed by Intel Corporation as the host programming environment for applications using the full range of Intel microcomputer and microcontroller chips. A superset of *PL/M*, which we call *PPL/M*, has been implemented as the system-level language for the *DADO* prototypes. *PPL/M* provides a set of facilities to specify operations to be performed by independent PE's in parallel.

Intel's *PL/M* language is a conventional block-oriented language providing a full range of data structures and high-level statements. The following two syntactic conventions have been added to *PL/M* for programming the SIMD mode of operation of *DADO*. The design of these constructs was influenced by the methods employed in specifying parallel computation in the *GLYPNIR* language [Lowrie et al. 1975] designed for the *ILLIAC IV* parallel processor. The *SLICE* attribute defines variables and procedures that are resident within each PE. The second addition is a syntactic construct, the *DO SIMD block*, which delimits *PPL/M* instructions broadcast to descendent SIMD PE's. (In the following definitions, optional syntactic constructs are represented within square brackets.)

The SLICE attribute:

DECLARE variable[(dimension)] type SLICE;

name: PROCEDURE[(params)] [type] SLICE;

Each declaration of a SLICEd variable will cause an allocation of space for the variable to occur within the SIMD RAM of each PE (the initial 1K portion). SLICEd procedures are automatically loaded within the 7K MIMD portion of RAM (by an operating system executive resident in *DADO*'s CP).

Within a *PPL/M* program, an assignment of a value to a SLICEd variable will cause the transfer to occur within each enabled SIMD PE concurrently. A constant appearing in the right hand side will be automatically broadcast to all enabled PE's. Thus, the statement

X=5;

where X is of type BYTE SLICE, will assign the value 5 to each occurrence of X in each enabled SIMD PE. (Thus, at times it is convenient to think of SLICEd variables as vectors which may be operated upon, in whole or in part, in parallel.) However, statements which operate upon SLICEd variables can only be specified within the bounds of a DO SIMD block.

DO SIMD block:

DO SIMD;
r-statement$_0$;
...
r-statement$_n$;
END;

The r-statement is restricted to be any *PL/M* statement *incorporating only SLICEd variables and constants*.

In addition to the full range of instructions available in *PPL/M*, a *DADO* PE in MIMD mode will have available to it a set of built-in functions to perform the basic tree communication operations, in addition to functions controlling the various modes of execution. The interested reader is referred to [Stolfo et al. 1982] for the details of these primitives, as well as a complete specification of the *PPL/M* language.

5. Applications

Thus far, 12 people have written *PPL/M* programs for *DADO*. The applications that have been written, at various stages of completion, include system-level diagnostics, numeric processing and AI applications. The largest share of our software effort, though, has concentrated on parallel implementations of various AI applications.

The most important of these is an algorithm for the parallel execution of production system programs. A restricted model of production systems has been implemented in *PPL/M* using this algorithm and is currently being tested.

In its simplest form, the algorithm operates in the following way:

1. By assigning a single rule to a unique PE at a fixed level within the tree (referred to as the

PM-level), executing in MIMD mode, each rule in the system is matched concurrently. Thus, the time to calculate the set of matching rules on each cycle is *independent of the number of productions in the system.*

2. By assigning a data item in Working Memory (WM) to a single PE below the PM-level executing in SIMD mode, WM is implemented as a true hardware *content-addressable memory.* Thus, the time required to match a single pattern element is *independent of the number of facts in WM.*

3. Lastly, the selection of a single rule for execution from the conflict set is also performed in parallel. Thus, the *logarithmic time lower bound* of comparing and selecting a single item from a collection of items is achievable on *DADO* as well.

This algorithm offers a number of advantages over the highly efficient RETE match algorithm reported by Forgy, while maintaining much of its inherentently efficient capabilities. We quote from [Forgy 1982]:

> ...Certainly the [RETE] algorithm should not be used for all match problems: its use is indicated only if the following three conditions are satisfied.
>
> - The patterns must be compilable [to more primitive match tests] ...
>
> - The objects must be constant. They cannot contain variables or other non-constants as patterns can.
>
> - The set of objects must change relatively slowly. Since the algorithm maintains state between cycles, it is inefficient in situations where most of the data changes on each cycle.

In its current form, the *DADO* algorithm does not provide a means to compile patterns into primitive match tests, although it does not directly exclude this possibility. However, the ability of a *DADO* PE to execute a rule independently of other PE's permits pattern matching tests common to several rules to be performed in parallel, as well as a more powerful pattern match operation: *unification.* Thus, data items within *DADO*'s WM may contain variables or other non-constants. (It is interesting to note that this capability forms the basis of the implementation of *PROLOG* on *DADO.*) Lastly, the *DADO* algorithm does not restrict the amount or scope of WM modifications, but rather permits large global changes to be made to WM very efficiently (by broadcasting such changes from the root PE). However, the *DADO* algorithm does not save state between cycles. Rather, in situations in which few WM changes are made on each cycle, the *DADO* algorithm recomputes much of its match results calculated on the previous cycle. We have recently discovered, though, that the basic *DADO* algorithm can be easily extended to directly implement this temporal redundancy.

Many problem solving tasks, though, are well suited to implementation using the RETE algorithm. Thus, we have also recently begun work on implementing the RETE match algorithm directly on *DADO.* It is too early in our investigations to make any precise claims about the performance of the RETE algorithm on *DADO.* However, since the data-flow discrimination network compiled by the RETE algorithm produces nodes with a maximum fan-in of two, it is our belief that the match network may be directly mapped onto the *DADO* tree structure permitting massive parallelism to be directly exploited. The comparative evaluation of the two approaches forms a major part of our current research activities.

Our plans include the completion of an interpreter for a more general version of production systems in the coming months. An improved algorithm for the rapid evaluation of *hierarchical production systems*, typified by *MYCIN*-like systems, is being investigated for implementation on *DADO* as well.

Lastly, we note the relationship of *PROLOG* and *DADO.* Since *PROLOG* may be considered as a special case of production systems, it is our belief that *DADO* can quite naturally support performance improvements of *PROLOG* programs over von Neumann implementations. Some interesting work in this direction has been reported in [Taylor et al. 1983].

By way of summary, it is our belief that *DADO* can in fact support the high-speed execution of a very large class of AI applications, in particular, the rapid execution of expert systems implemented in production system form. Coupled with an efficient implementation in VLSI technology, the large-scale parallelism achievable on *DADO* will indeed provide significant performance improvements over conventional machines at moderately low cost. Indeed, our preliminary statistics suggest that *DADO2* is expected to execute *R1/XCON*, the Digital Equipment Corporation VAX configuration program reported by McDermott [1981], at an average rate in excess of *150 production system cycles per second!* Presently, *R1/XCON* executes on a VAX 11/780, at a rate of 2 to 600 production system cycles per minute. It is interesting to note further that this larger prototype will be comparable in hardware complexity and cost to the DEC VAX 11/750, a smaller, slower and much less expensive version of the VAX 11/780.

6. Conclusion and Future Research

A large part of our work continues to involve the analytical investigation of new parallel algorithms and languages for AI applications. Several researchers are actively investigating methods for the rapid execution of *PROLOG* programs. A large share of our efforts, though, are devoted to the hardware design and implementation of a larger experimental device. Although adequate for further development of the software base for *DADO*, the 15-element system is too limited in storage capacity and processing power to demonstrate a significant performance improvement in the execution of AI systems. Thus, using the hardware design of the 15-element system, we are currently implementing *DADO2*, comprising 1023 processing elements. *DADO2* will incorporate a custom VLSI chip, currently being designed at Columbia University, to perform the most basic communicaton functions in combinational logic.

Our future plans include the demonstration of the *DADO2* prototype using several existing large-scale expert systems which use the production system paradigm. Digital Equipment Corporation has expressed an interest in supplying a copy of *R1/XCON* for implementation on *DADO2*. Bell Laboratories has also expressed a willingness to supply a copy of *ACE*, an expert system that has been developed to perform telephone cable maintenance [Stolfo and Vesonder 1982]. Other systems are being actively sought from other sources in the artificial intelligence community.

REFERENCES

Fahlman, S. E., *The Hashnet Interconnection Scheme*, Technical Report 125, Department of Computer Science, Carnegie-Mellon University, 1979.

Forgy, C. L., "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *AI* 19, 1982.

Hillis, D., *The Connection Machine*, Technical Report, Department of Computer Science, Massachussetts Institute of Technology, 1982

Intel Corporation, *PL/M-51 Users's Guide for the 8051 Based Development System*, Order Number 121966, 1982.

Lesser, V. and D. Corkill, "The application of Artificial Intelligence to cooperative distributed processing", In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan, 1979.

Lowrie, D., T. Layman, D. Daer and J. M. Randal, "*GLYPNIR*-A Programming Language for *ILLIAC IV*", *Comm. ACM*, 18 3, March, 1975.

McDermott, J., "*R1*: The Formative Years", *AI Magazine* 2:21-29, 1981.

Nilsson, N., *Fundamental Principles of Artificial Intelligence*, Tioga Press, Menlo Park, California, 1980

Shaw, D. E., S. J. Stolfo, H. Ibrahim, B. Hillyer, G. Wiederhold and J. A. Andrews, "The NON-VON Database Machine: A Brief Overview", *Database Engineering* 4(2), 1981.

Stolfo, S. J., D. Miranker and D. E. Shaw, *Programming the DADO Machine: An Introduction to PPL/M*, Technical Report, Department of Computer Science, Columbia University, 1982.

Stolfo, S. J. and D. E. Shaw, "DADO: A Tree-structured Machine Architecture for Production Systems", In *Proceedings National Conference on Artificial Intelligence*, Carnegie-Mellon University and University of Pittsburgh, August, 1982.

Stolfo, S. J. and G. T. Vesonder, *ACE: An Expert System Supporting Analysis and Management Decision Making*, Technical Report, Department of Computer Science, Columbia University, 1982.

Taylor, S., C. Maio, S. J. Stolfo and D. E. Shaw, *PROLOG on the DADO Machine: A Parallel System for High-Speed Logic Programming*, Technical Report, Department of Computer Science, Columbia University, 1983.