CUCS-122-84

DADO: A PARALLEL PROCESSOR FOR EXPERT SYSTEMS

SALVATORE J. STOLFO
AND
DANIEL P. MIRANKER

CUCS-122-84

DADO: A Parallel Processor for Expert Systems*

Salvatore J. Stolfo
and
Daniel P. Miranker
Department of Computer Science
Columbia University
New York City, N. Y. 10027

Abstract -- DADO is a parallel, tree-structured machine designed to provide significant performance improvements in the execution of large expert systems implemented in production system form. A full-scale version of the DADO machine would comprise a large (on the order of a hundred thousand) set of processing elements (PE's), each containing its own processor, a small amount (16K bytes, in the current prototype design) of local random access memory, and a specialized I/O switch. The PE's are interconnected to form a complete binary tree.

This paper describes the application domain of the DADO machine and the rationale for its design. We then focus on the machine architecture and detail the hardware design of a moderately large prototype comprising 1023 microprocessors currently under development at Columbia University. We conclude with very encouraging performance statistics recently calculated from an analysis of extensive simulations of the system.

Introduction

Due to the dramatic increase in computing power and the concomitant decrease in computing cost occurring over the last decade, many researchers are attempting to design computing systems to solve complicated problems or execute tasks which have in the past been performed by human experts. The focus of *Knowledge Engineering* is the construction of such complex, knowledge-based expert computing systems.

In general, knowledge-based expert systems are Artificial Intelligence (AI) problem-solving programs designed to operate in narrow "real-world" domains, performing tasks with the same competence as a skilled human expert. Illucidation of unknown chemical compounds [3], medical diagnosis [23], mineral exploration [4] and telephone cable maintenance [30] are just a few examples. The heart of these systems is a *knowledge base*, a large collection of facts, definitions, procedures and heuristic "rules of thumb", *acquired directly from a human expert*. The knowledge engineer is an intermediary between the expert and the system who extracts, formalizes, represents, and tests the relevant knowledge within a computer program.

*This research has been supported by the Defense Advanced Research Projects Agency through contract N00039-82-C-0427, as well as grants from Intel, Digital Equipment, Hewlett-Packard, Valid Logic Systems, AT&T Bell Laboratories and IBM Corporations and the New York State Science and Technology Foundation. We gratefully acknowledge their support.

Just as robotics and CAD/CAM technologies offer the potential for higher productivity in the "blue-collar" work force, it appears that AI expert systems will offer the same productivity increase in the "white-collar" work force. As a result, Knowledge Engineering has attracted considerable attention from government and industry for research and development of this emerging technology. However, 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. Much of the research in AI has focused on the problem of representing and organizing knowledge, but little attention has been paid to specialized machine architectures supporting problem-solving programs.

DADO is a large-scale parallel machine designed to support the rapid execution of expert systems, as well as multiple, independent systems. In the following sections we present an overview of DADO's application domain as well as the rationale for its design. We then detail the hardware design of the *DADO2* prototype, currently under construction at Columbia University, consisting of *1023 microprocessors*. We conclude with a presentation of performance statistics recently calculated from extensive simulations of the system, and an overview of the software systems implemented to date. Based on our studies, a full scale version of *DADO* comprising many thousands of processing elements will, in our opinion, be technically and economically feasible in the near future.

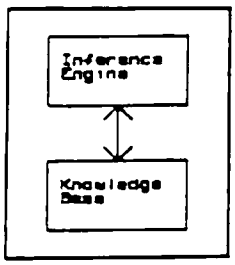
Expert Systems

Current Technology. Knowledge-based expert systems have been constructed, typically, from two loosely coupled modules, collectively forming the *problem-solving engine* (see Figure 1). The *knowledge base* contains all of the relevant domain-specific information permitting the program to behave as a specialized, intelligent problem-solver. Expert systems contrast greatly with the earlier general-purpose AI problem-solvers which were typically implemented without a specific application in mind. One of the key differences is the large amounts of problem-specific knowledge encoded within present-day systems.

Much of the research in AI has concentrated on effective methods for representing and operationalizing human experiential domain knowledge. The representations that have been proposed have taken a variety of forms including purely declarative-based logical formalisms, "highly-stylized" rules or productions, and

structured generalization hierarchies commonly referred to as semantic nets and frames. Many knowledge bases have been implemented in rule form, to be detailed shortly.

Figure 1: Organization of a Problem-Solving Engine.



The *inference engine* is that component of the system which *controls* the deductive process: it implements the most appropriate strategy, or *reasoning* process for the problem at hand. The earliest AI problem-solvers were implemented with an iterative branching technique searching a large combinatorial space of problem states. Heuristic knowledge, applied within a static control structure, was introduced to limit the search process while attempting to guarantee the successful formation of solutions. In contrast, state-of-the-art expert systems separate the control strategy from an inflexible program, and deposit it in the knowledge base along with the rest of the domain-specific knowledge. Thus, the problem-solving strategy becomes domain-dependent, and is responsible to a large extent for the good performance exhibited by today's systems. However, a great deal of this kind of knowledge is necessary to achieve highly competent performance.

Within a great number of existing expert system programs, the corpus of knowledge about the problem domain is embodied by a *Production System* program. As has been reported by several researchers, production system representation schemes appear well suited to the organization and implementation of knowledge-based software. Rule-based systems provide a convenient means for human experts to explicate their knowledge, and are easily implemented and readily modified and extended. Thus, it is the ease with which rules can be acquired and explained that makes production systems so attractive.

Production Systems. In general, a *Production System* [6, 17, 18, and 19] is defined by a set of rules, or *productions*, which form the *Production Memory*(PM), together with a database of assertions, called the *Working Memory*(WM). Each production consists of a conjunction of *pattern elements*, called the *left-hand side* (LHS) of the rule, along with a set of actions called the *right-hand side* (RHS). The RHS specifies information that is to be added to (asserted) or removed from WM when the LHS successfully matches against the contents of WM. An example production, borrowed from the blocks world, is illustrated in Figure 2.

In operation, the production system repeatedly executes the following cycle of operations:

1. *Match*: For each rule, determine whether the LHS matches the current environment of WM. All matching instances of the rules are collected in the *conflict set of rules*.

2. *Select*: Choose exactly one of the matching rules according to some predefined criterion.

3. *Act*: Add to or delete from WM all assertions specified in the RHS of the selected rule or perform some operation.

Figure 2: An Example Production.

```
(Goal (Clear-top-of Block))
(Isa =x Block)
(On-top-of =y =x)
(Isa =y Block)  ->
                delete(On-top-of =y =x)
                assert(On-top-of =y Table)

If the goal is to clear the top of a block,
  and there is a block (=x)
  covered by something (=y)
  which is also a block,
            then
             remove the fact that =y is on =x
             and assert that =y is on top of the table.
```

During the selection phase of production system execution, a typical interpreter provides *conflict resolution strategies* based on the *recency* of matched data in WM, as well as syntactic discrimination. Rules matching data elements that were more recently inserted in WM are preferred, with ties decided in favor of rules that are more specific (i.e., have more constants) than others.

Why a specialized PS architecture? One problem facing expert systems technology is efficiency. It should be evident from the above description that large PS programs would spend most of their time executing the match phase requiring an enormous number of primitive symbol manipulation tasks. (Indeed, Forgy [6] notes that some PS interpreters spend 90% of their time in the match phase.) Hence, as this technology is ambitiously applied to larger and more complex problems, the size and concomitant slow speed of execution of production system programs, *with large rule bases*, on conventional machines will most likely doom such attempts to failure. The *R1* program [13], designed to configure Digital Equipment Corporation VAX computers, provides a convincing illustration.

In its current form, *R1* contains approximately 2500 rules operating on a WM containing several hundred data items, describing a partially configured VAX. Running on a DEC VAX 11/780 computer and implemented in OPS5 [8], a highly efficient production system language, *R1* executes from 2 to 600 production system cycles per minute. Configuring an entire VAX system requires a considerable amount of computing time on a moderately large and expensive computer. The performance of such systems

will quickly worsen as experts are designed with not only one to two thousand rules, but perhaps with *tens of thousands* of rules. Indeed, several such large-scale systems are currently under development at various research centers. Statistics are difficult to calculate in the absence of specific empirical data, but it is conceivable that such large systems may require an unacceptable amount of computing time for a medium size conventional computer to execute a single cycle of production system execution! Thus, we consider the design and implementation of a specialized *production system machine* to warrant serious attention by parallel architects and VLSI designers.

Much of the experimental research conducted to date on specialized hardware for AI applications has focused on the realization of high-performance, cleverly designed, but for the most part, architecturally conventional machines. (MIT's LISP Machine exemplifies this approach.) Such machines, while quite possibly of great practical interest to the research community, make no attempt to employ hardware parallelism on the massive scale characteristic of our own work.

Thus, simply stated, the goal of the DADO machine project is the design and implementation of a *cost effective* high performance *rule processor*, based on large-scale parallel processing, capable of rapidly executing a production system cycle for very large rule bases. The essence of our approach is to execute a very large number of pattern matching operations on concurrent hardware, thus substantially accelerating the match phase. Our goals do not include the design of a high-speed parallel processor capable of a fruitless parallel search through a combinatorial solution space.

A small (15 processor) prototype of the machine, constructed at Columbia University from components supplied by Intel Corporation, has been operational since April 1983. Based on our experiences with constructing this small prototype, we believe a larger DADO prototype, comprising 1023 processors, to be technically and economically feasible for implementation using current technology. We believe that this larger 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.

The DADO Machine

The System Architecture. DADO is a fine-grain, parallel machine where processing and memory are extensively intermingled. A full-scale production version of the DADO machine would comprise a very large (on the order of a hundred thousand) set of *processing elements* (PE's), each containing its own processor, a small amount (16K bytes, in the current design of the prototype version) of local random access memory (RAM), and a specialized I/O switch. The PE's are interconnected to form a *complete binary tree* (see Figure 3).

Within the DADO machine, each PE is capable of executing in either of two modes under the control of run-time software. In the first, which we will call *SIMD mode* (for single instruction stream, multiple data stream [5]), the PE executes instructions broadcast by some ancestor PE within the tree. (SIMD typically refers to a single stream of "machine-level" instructions. Within DADO, on the other hand, SIMD is generalized to mean a single stream of remote procedure invokation instructions. Thus, DADO makes more effective use of its communication bus by broadcasting more "meaningful" instructions.) In the second, which will be referred to as *MIMD mode* (for multiple instruction stream, multiple data stream), each PE executes instructions stored in its own local RAM, independently of the other PE's. A single conventional coprocessor, 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. This flexible architectural design supports *multiple-SIMD* execution (MSIMD), as, for example, [24], but on a much larger scale. Thus, the machine may be logically divided into distinct partitions, each executing a distinct task, and is the primary source of DADO's speed in executing a large number of primitive pattern matching operations concurrently.

The DADO I/O switch, which will be implemented in semi-custom gate array technology and incorporated within the 1023 processing element version of the machine, has been designed to support rapid global communication. In addition, a specialized combinational 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, a process we call resolving. Currently, the 15 processing element version of DADO performs these operations in firmware embodied in its off-the-shelf components.

The Binary Tree Topology. In our initial work, several alternative parallel machine architectures were studied to determine a suitable organization of a special-purpose production system machine. High-speed algorithms for the parallel execution of production system programs were developed for the perfect shuffle [21] and binary tree machine architectures [1]. Forgy [7] proposed an interesting use of the mesh-connected ILLIAC IV machine [12] for the parallel execution of production systems, but recognized that his approach failed to find all matching rules in certain circumstances. Of these architectures, the binary tree organization was chosen for implementation. For the present paper we summarize these reasons as follows:

- Binary trees are efficiently implemented in VLSI technology:
    - Using the well known "Hyper-H" embedding (see [2]), binary trees can be embedded in the plane in an amount of area proportional to the number of processors. Thus, as VLSI continues scaling downward, higher processor densities can be achieved.
    - A design for a single chip type, first reported by Leiserson [11], embeds both a complete binary subtree and one additional PE, which can be used to implement an arbitrarily large binary tree. Thus, binary tree machines have a very low number of distinct integrated parts.

* Pin-out on the Leiserson chip remains constant for any number of embedded PE's.

* The Leiserson chip used with a simple recursive construction scheme produces printed circuit board designs that make optimal use of available area. This single printed circuit board design is suitable for implementing an arbitrarily large binary tree.

- Broadcasting data to a large number of recipients is handled efficiently by tree structures.

- Most importantly, the binary tree topology is a natural fit for production system programs.

We note that binary trees do have certain limitations of practical importance. Although broadcasting a small amount of information to a large number of recipients is efficiently handled by binary trees, the converse is, in general, unfortunately not true. That is, for certain computational tasks (permutation of data within the tree, for example) the effective bandwidth of communication is restricted by the top of the tree. Fortunately, as we shall see shortly, this "binary tree bottleneck" does not arise in the execution of production systems.

Production System execution. In our earlier work, extensive theoretical analyses and software simulations of a high-speed algorithm for production system execution on *DADO* was completed and reported in [25]. In its *simplest* form, the algorithm operates in the following way:

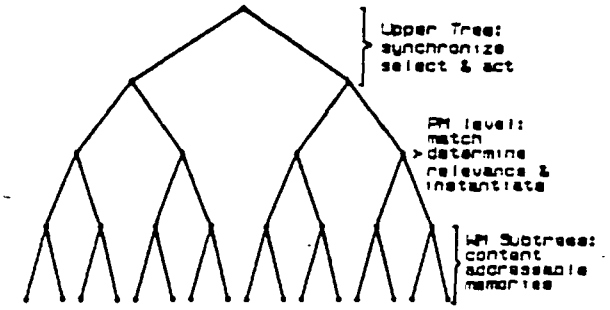
1. By assigning a single rule to each PE, executing in MIMD mode, at a (logically) fixed level within the tree, each rule in PM is matched concurrently. (This fixed level within the tree is referred to as the PM-level, see Figure 3.) Thus, the time to calculate the conflict set of rules on each cycle is independent of the number of productions in the system. Variations of this approach allow for multiple rules to be located at a PM-level PE, thus increasing the time to match a modest degree.

2. By assigning a data item in WM to a single PE executing in SIMD mode, lying below the PM-level, WM is implemented as a true hardware *content-addressable memory*. Thus, the time required to match a single pattern element in the LHS of a rule is independent of the number of facts in WM. (In a manner similar to production storage, more effective use of the WM PE's is made by allowing several WM elements to be present. The WM elements stored at a single PE, however, are "disjoint" in the sense that they may match different condition elements in the LHS of a rule.)

3. The selection of a single rule for execution from the conflict set is also performed in parallel by a logarithmic time binary tree selection executed above the PM-level. Thus, the logarithmic time lower bound of comparing and selecting a single item from a collection of items is achievable on *DADO* as well.

4. Lastly, the RHS actions specified by the selected rule are broadcast to all PM-level PE's which update their respective WM-subtrees in parallel.

Figure 3: Functional Division of the DADO Tree.



A comparative evaluation of this algorithm with various allocation schemes has been reported elsewhere (see [10, 15, 25, 27, and 28]). It should be noted that although the running time of the basic algorithm is shown to be insensitive to the size of PM and WM, in practice a fixed size machine may not, in general, attain these lower bound results. Thus, in situations where WM and PM are too large to be conveniently distributed in the manner discussed above for a machine of fixed size, some performance degradation will result.

For example, the second DADO prototype will consist of 1023 PE's and is expected to be logically divided with a PM-level consisting of 32 PE's, each rooting a WM-subtree with 31 PE's. To execute a 2500 rule system such as R1 will require partitioning ~75 rules to each PM-level PE. It would appear that the time to match would depend on 75 rules rather than 2500 for this example. However, recent statistics reported indicate that never more than ~30 rules are active on each cycle of execution of R1. Hence, with a suitable partitioning of rules, no more than 1 rule would be processed by each PM-level PE in our example configuration, thus attaining a match time independent of 2500 rules. Note, though, that each PM-level PE can access 31 WM elements in parallel. Thus, in total, 32 X 31 or ~1000 WM elements would be accessed at any one point in time. In the case where a single rule might require access to more than 31 WM elements at a time, performance will degrade gracefully. Hence, 31 elements can be accessed by a single PM-level PE in one time unit, 62 in two time units, 93 in three, etc.

Recently, we have completed a number of reports which detail five related algorithms for the parallel execution of PS programs to account for various differences in PS programs. As noted, some PS programs, (R1, for example) may not have a high degree of "production-level parallelism". That is, on each cycle only a relatively small number of productions may have satisfied LHS's. Other PS programs may have a high degree of production-

level parallelism. Many other variations are possible which lead to a variety of related algorithms which attempt to maximize system performance by integrating various rule partitioning schemes with clever "state saving" schemes. The details of the various methods are beyond the scope of this paper, and thus the reader is encouraged to see [15] and [28]. Studies of such situations have been made and the projections of possible performance degradation are summarized in a later section of this paper.

Although analytical studies and software implementations are primary tasks of the *DADO* project, our current efforts have focused on the construction of hardware. Many parallel computing devices have been proposed in the literature, however, often such devices are constructed only on paper. Many scientific and engineering problems remain undetected until an actual device is constructed and experimentally evaluated. Thus, we are actively building a large prototype consisting of 1023 Intel 8751 microcomputer chips. A small 15 PE version of *DADO* is currently operational at Columbia University acting as a development system for the software base of the larger prototype. In the remainder of this paper we concentrate on the details of the hardware for these prototypes as well as the software systems that have been implemented thus far.

The DADO Prototypes

Physical Characteristics. A 15-element *DADO1* prototype, constructed from (partially) donated parts supplied by Intel Corporation, has been operational since April 25, 1983. The two wire-wrap board system, housed in a chassis roughly the size of an IBM PC, is clocked at 3.5 megahertz producing 4 million instructions per second (MIPS)[16]. (The effective usable MIPS is considerably less due to the significant overhead incurred in interprocessor communication. For each byte quantity communicated through the system, 12 machine instructions are consumed at each level in the tree while executing an asynchronous, 4-cycle handshake protocol.) DADO1 contains 124K bytes of user random access storage and 60K bytes of read only memory. A much larger version, *DADO2*, is currently under construction which will incorporate 1023 PE's constructed from two commercially available Intel chips and one semi-custom gate array chip (to be fabricated by LSI Logic). DADO1 does not provide enormous computational resources. Rather, it is viewed as the development system for the software base of DADO2, and is not expected to demonstrate a significant improvement in the speed of execution of a production system application.

DADO2 will be implemented with 32 printed circuit boards housed in an IBM Series I cabinet (donated by IBM Corporation). A DEC VAX 11/750 (partially donated by DEC Corporation) serves as DADO2's coprocessor (although an HP 9836 workstation may be used as well) and is the only device a user of DADO2 will see. Thus, DADO2 is considered a transparent back-end processor to the VAX 11/750.

The DADO2 system will have roughly the same hardware complexity as a VAX 11/750 system, and if amortized over 12 units will cost in the range of 70 to 90 thousand dollars to construct considering 1982 market retail costs. The DADO2 semi-custom I/O chip is planned for implementation in gate array technology and will allow DADO2 to be clocked at 12 megahertz, the full speed of the Intel chips. The average machine instruction cycle time is 1.8 microseconds, producing a system with a raw computational throughput of roughly 570 million instructions per second. We note that little of this computational resource is wasted in communication overhead as in the DADO1 machine.

The Prototype Processing Element. Each PE in the 15-element DADO1 prototype system incorporates an Intel 8751 microcomputer chip, serving as the processor, and an 8K X 8 Intel 2186 RAM chip, serving as the local memory. DADO2 will incorporate a slightly modified PE. The Intel 2187, which is fully compatible with but faster than an Intel 2186, replaces the DADO1 RAM chip allowing the processor to be clocked at its fastest speed. Two such chips will be used (with a 16K X 1 chip for parity), increasing the PE storage capacity to 16K bytes. Further, the custom I/O chip will contain memory support circuitry and thus also replaces several additional gates employed in DADO1.

Although the original version of DADO had been designed to incorporate a 2K byte RAM within each PE, a 16K byte 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. In addition, this extra "breathing room" within each PE allows for experimentation with various special operations that may be incorporated in the full version of the machine in combinational circuitry, as well as affording the opportunity to critically evaluate other proposed (tree-structured) parallel architectures through software simulation.

It is worth noting though that the proper choice of "grain size" is an interesting open question. That is, through experimental evaluation we hope to determine the size of RAM for each PE, chosen against the number of such elements for a fixed hardware complexity, appropriate for the widest range of production system applications. Thus, future versions of DADO may consist of a number of PE's each containing an amount of RAM significantly larger or smaller than implemented in the current prototype systems.

The Intel 8751 is a moderately powerful 8-bit microcomputer incorporating a 4K erasable programmable read only memory (EPROM), and a 256-byte RAM on a single silicon chip. One of the key characteristics of the 8751 processor is its I/O capability. The 4 parallel, 8-bit ports provided in a 40 pin package has contributed substantially to the ease of implementing a binary tree interconnection between processors. Indeed, DADO1 was implemented within 4 months of delivery of the hardware components. Figure 4 illustrates the DADO1 prototype PE, while figure 5 illustrates DADO2's PE.

Note that the same processor connections exist in the DADO2 PE design as those appearing in the DADO1 design. If in the unlikely event that the planned I/O chip does not function properly, DADO2 will thus remain operational, but will not run as fast as envisaged. Since the DADO1 hardware to date has remained operable, we are convinced that the fully upward compatible DADO2 PE design ensures the successful operation of a 1023 PE version of the machine.

In DADO1 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 envisaged for the larger version of the machine is simulated in the

Figure 4: The DADO1 Prototype Processing Element.

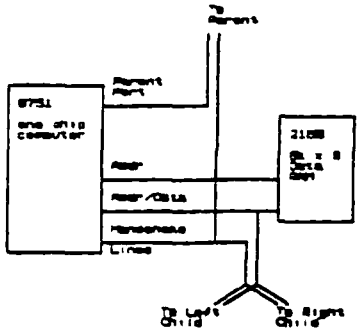
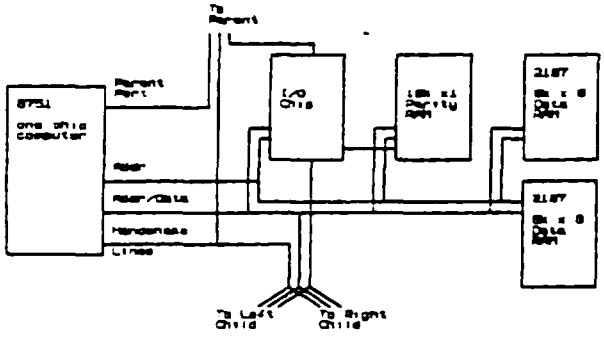


Figure 5: The DADO2 Prototype Processing Element.



smaller version by a short sequential computation. As noted, the 1023 element prototype would be capable of executing in excess of 570 MIPS. Although pipelined communication is employed in the DADO1 kernel design, it is expected that fewer MIPS would be achieved on DADO2 without the I/O chip, as detailed in the following section. Thus, the design and implementation of a custom I/O chip forms a major part of our current hardware research activities.

It should be noted that, in keeping with our principles of "low-cost performance," we have selected a processor technology one generation behind existing available microcomputer technology. For example, DADO2 could have been designed with 1023 Motorola 68000 processors or Intel 80286 chips. Instead, we have chosen a relatively slow technology to limit the number of chips for each PE, as well as to demonstrate our most important architectural principals in a cost effective manner.

Furthermore, since the Intel 8751 does not press current VLSI technology to its limits, it is surely within the realm of feasibility to implement a DADO2 PE on a single silicon chip. Thus, although DADO2 may appear impressive (an inexpensive, compact system with a thousand computers executing roughly 600 million instructions per second) its design is very conservative and probably at least an order of magnitude less powerful than a similar device using faster technology. It is our conjecture though that the machine will be practical and useful and many of its limitations will be ameliorated as VLSI continues its downward trend in scaling. (DADO3 may serve to prove this conjecture.)

Performance Evaluation of DADO2.

Design Alternatives. Much of the available computing power in the *DADO1* prototype is consumed by firmware executing a four cycle handshake communication protocol. For this reason we investigated the tradeoffs involved with adding a specialized I/O circuit to each PE to handle global communication in *DADO2*. The current I/O circuit design provides the means to broadcast a byte to all PE's in the tree in less than one Intel 8751 instruction cycle. This efficiency gain does not come free. The I/O circuit increases a PE's component count as well as the total area on a printed circuit board for the system. To decide this issue, we investigated the relative performance of a machine design incorporating the I/O circuit and a design without the I/O circuit, using the available area for additional PE's.

A second but orthogonal issue for the machine design is whether or not it is worthwhile to buffer the instruction stream broadcast to PE's executing in SIMD mode. In a typical SIMD machine a control processor issues a stream of machine level instructions that are executed synchronously in lock step by all of the slave processors in the array. *DADO* is different. Since each PE of *DADO* is a fully capable computer, and communication between PE's is generally expensive, we wish to make an instruction as "meaningful" as possible. What is communicated as an instruction in *DADO* is usually a pointer to a procedure, stored locally in each slave PE. Primitive SIMD *DADO* instructions are in fact parallel procedure calls and may be viewed as macro instructions.

For example, a common instruction that will be executed by a *DADO* PE is "MATCH(pattern)", where MATCH is a generalized pattern matching routine local to each processor.

Transmitting pointers to procedures makes effective use of communication links but introduces a difficult problem. A procedure may behave differently depending on the local data. Thus, the same macro instruction may require different amounts of processing time in each PE. In such a device either the PE's must synchronize on every instruction, and therefore potentially lay idle while the slowest PE finishes, or the PE's must be able to buffer the instruction stream to possibly achieve better utilization. However, buffering the instructions requires overhead and may in fact decrease the overall performance.

Evaluation Method. To resolve these two design issues the *DADO* instruction stream was characterized by studying the code implementing the match phase of the *DADO* production system algorithm, (roughly 10 pages of PPL/M, detailed in the following section). Queuing models were developed for each configuration representing the 4 possible combinations: a DADO PE with and without the I/O circuit, and with and without buffering. The four models were simulated using the IBM Research Queuing Network Simulation package, RESQ2, [20]. The package has a number of

very powerful simulation primitives including generation of job streams with a variety of distributions times, active queues with a variety of queueing service disciplines as well as mechanisms to provide flow control. Complete details of this study can be found in [14].

Evaluation Results. Figure 6 summarizes the relative throughput of the four configurations working on a problem typical of the size we expect a 1023 node *DADO* to handle: 1000 productions and 1000 working memory elements (although for certain PS programs, R1, for example, we will be able to implement nearly 2500 production rules). The simulations show that the I/O circuit can be expected to nearly double the performance of the *DADO* machine. However, the overhead associated with buffering causes a decrease in performance of 27 and 20 percent in configurations with and without the I/O circuit, respectively.

Figure 6: Relative Performance of Four PE Configurations.

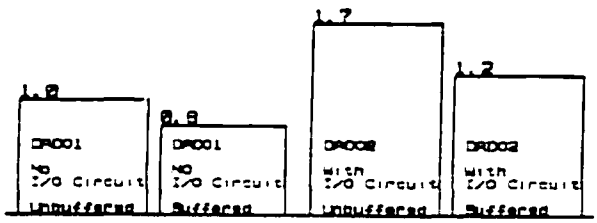


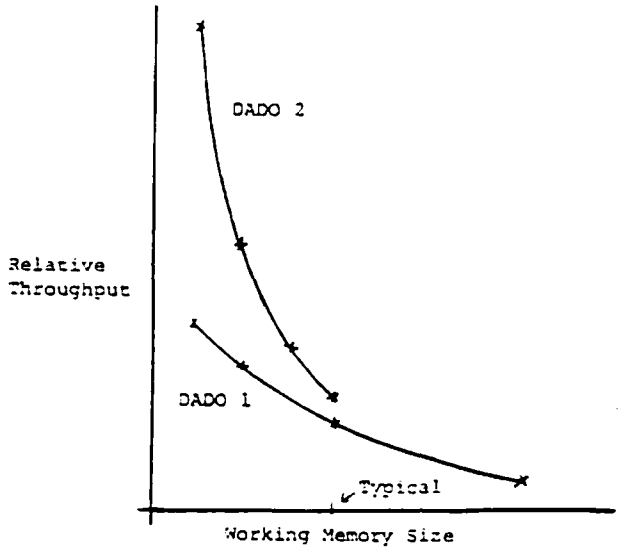
Figure 7 is a comparison of a 5 level *DADO* subtree (comprising 31 PE's) without the I/O circuit, and a 4 level *DADO* subtree, (comprising 15 PE's) with the I/O circuit. The x-axis represents a rough approximation of the number of WM data elements in the system. The graph shows that for a typical size problem a 9 level deep *DADO2* with the I/O circuit will outperform a 10 level deep *DADO1* without the I/O circuit by roughly 15 percent. However, the smaller machine's performance degrades faster than that of the larger machine. The simulations indicate for problems larger than those we anticipate it is worthwhile to dispense with the I/O circuit in favor of additional PE's.

Programming DADO

PL/M, [9] 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 [12] designed for the *ILLIAC IV* parallel processor. The *SLICE* attribute defines variables and procedures

Figure 7: Performance Comparison of DADO1 and DADO2 on Variable Size Working Memory.



that are resident within each PE. The second addition is a syntactic construct, the *DO SIMD block*, which delimits *PPL/M* instructions broadcast to descendant 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 each PE. SLICEd procedures are automatically loaded within the RAM of each PE by an operating system executive resident in *DADO*'s coprocessor.

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-statement0;
  ...
 r-statementn;
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.

Direct hardware support is provided by the semi-custom I/O chip for each of the global communication functions: BROADCAST, REPORT and RESOLVE, other communication primitives are implemented by firmware embedded in the processor EPROM. The interested reader is referred to [26] for the details of these primitives, as well as a complete specification of the *PPL/M* language.

The RESOLVE instruction recently redesigned from studying DADO1's behavior deserves special mention here. The RESOLVE instruction is used in practice to disable all but a single PE, chosen from among a specified set of PE's. In DADO1, first a SLICEd variable is set to one in all PE's to be included in the candidate set. The RESOLVE instruction is then issued by a PE executing in MIMD mode, causing all but one of the flags in descendant PE's, executing in SIMD mode, to be changed to zero. (Upon executing a RESOLVE instruction, one of the inputs to the MIMD PE will become high if at least one candidate was found in the tree, and low if the candidate set was found to be empty. This condition code is stored in a SLICEd variable, which exists within the MIMD PE.) By issuing an assignment statement, all but the single, chosen PE may be disabled, and a sequence of instructions may be executed on the chosen PE alone. In particular, data from the chosen PE may be communicated to the MIMD PE through a sequence of REPORT commands.

In DADO1, the RESOLVE function is implemented using special sequential code, embedded within the EPROM, that propagates a series of "kill" signals in parallel from all candidate PE's to all (higher-numbered) PE's in the tree. In DADO2, the RESOLVE operation has been generalized to operate on 8-bit data, producing the *maximum* value stored in some candidate PE. Repeated use of this max-RESOLVE function allows for the very rapid selection of multiple byte data. This circuit has proven very useful for a number of DADO algorithms which made use of the tree neighbor communication instructions primarily for ordering data within the tree. The use of the high-speed max-RESOLVE often obviates the need for such communication instructions. Consequently, the view of DADO as a binary tree architecture has become, fortuitously, nearly transparent in most of the algorithms written for DADO thus far.

Conclusion

The largest share of our software effort has concentrated on parallel implementations of various AI applications. The most important of these is an interpreter for the parallel execution of production system programs. A restricted model of production systems has been implemented in *PPL/M* and is currently being tested. Our plans include the completion of an interpreter for a more general version of production systems in the coming months.

We have also become very interested recently in *PROLOG*. 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 conventional implementations. Some interesting work in this direction has been reported in [31].

Lastly, we note the relationship of *LISP* to *DADO*. Part of our work has concentrated on providing *LISP* with additional parallel processing primitives akin to those employed in *PPL/M*. We have come to use PSL LISP this purpose due to its relative ease in porting to a new processor.

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 specifically expert systems implemented in rule form. Coupled with an efficient implementation in VLSI technology, the large-scale parallelism achievable on *DADO* will indeed provide significant performance improvements over von Neumann machines. Indeed, our preliminary statistics suggest that the 1023 PE version of *DADO* is expected to execute *R1*, for example, at an average rate in excess of *85 production system cycles per second!* Present statistics for a reimplementation of R1 on a VAX 11/780 project a performance of 30-50 cycles per second. It is interesting to note further that the DADO2 prototype will be comparable in hardware complexity to the DEC VAX 11/750, a smaller, slower and much less expensive version of the VAX 780 used presently to execute *R1*. Hence, DADO2's parallelism achieves a 50% performance improvement over a machine roughly six times its size.

References

[1] Browning, S., "Hierarchically organized machines," In Mead and Conway (Eds.), *Introduction to VLSI Systems*, 1978.

[2] Browning, S., *The Tree Machine: A Highly Concurrent Computing Environment*, Ph.D. Thesis, California Institute of Technology, 1980.

[3] Buchanan, B. G. and Feigenbaum, E. A., "DENDRAL and Meta-DENDRAL: Their applications dimension," *Artificial Intelligence*, 11:5-24, 1978.

[4] Duda, R., Gashnig, J. and Hart, P.E., "Model design in the PROSPECTOR consultant system for mineral exploration," In D. Michie (Ed.), *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press, 1979.

[5] Flynn, M. J., "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, 1972.

[6] Forgy, C. L., *On the Efficient Implementation of Production Systems*, Ph.D. Thesis, Carnegie-Mellon University, 1979.

[7] Forgy, C. L., "A note on production systems and ILLIAC IV," Technical Report 130, Department of Computer Science,

Carnegie-Mellon University, 1980.

[8] Forgy, C. L., "RETE: A fast algorithm for the many pattern/many object pattern problem," *Artificial Intelligence Journal*, 1982.

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

[10] Ishida, T. and S. J. Stolfo, "Simultaneous firing of production rules on tree-structured machines," Technical Report, Department of Computer Science, Columbia University, 1984. (Submitted to *Int. Conf. Fifth Generation Computer Systems.*)

[11] Leiserson, C. E., *Area-Efficient VLSI Computation*, Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, 1981.

[12] Lowrie, D. D., T. Layman, D. Daer and J. M. Randal, "GLYPNIR-A programming language for ILLIAC IV," *Comm. ACM*, 18-3, 1975.

[13] McDermott, J., "R1: The formative years," *AI Magazine* 2:21-29, 1981.

[14] Miranker, D. P., "The performance analysis of four competing DADO PE configurations," Technical Report, Department of Computer Science, Columbia University, 1983.

[15] Miranker, D. P., "Performance estimates for the DADO machine: A comparison of TREAT and RETE," Technical Report, Department of Computer Science, Columbia University, 1984. (Submitted to *Int. Conf. on Fifth Generation Computer Systems.*)

[16] Miranker, D. P., "The system-level design of the DADO1 prototype," (in preparation).

[17] Newell, A., "Production systems: models of control structures," In W. Chase (editor), *Visual Information Processing*, Academic Press, 1973.

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

[19] Rychener, M., *Production Systems as a Programming Language for Artificial Intelligence Research*, Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, 1976

[20] Sauer, Charles H., Macnair, Edward A., Kurose, James F. "The research queueing package, CMS User's Guide," Technical Report RA 139 #41127, IBM Research Division, 1982.

[21] Schwartz, J. T., "Ultracomputers," *ACM Transactions on Programming Languages and Systems* 3(1), 1980.

[22] Shaw, D. E., "The NON-VON supercomputer," Technical Report, Department of Computer Science, Columbia University, 1982.

[23] Shortliffe, E. H., *Computer-Based Medical Consultations: MYCIN*, New York: American Elsevier, 1976.

[24] Siegel, H. J., L. J. Siegel, F. C. Kemmerer, P. T. Mueller, H. E. Smolky and D. S. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Transactions on Computers*, 1981.

[25] Stolfo, S. J. and D. E. Shaw, "DADO: A tree-structured machine architecture for production systems," *Proc. National Conference on Artificial Intelligence*, Carnegie-Mellon University and University of Pittsburgh, 1982.

[26] Stolfo, S. J., D. Miranker and M. Lerner, "PPL/M: The system level language for programming the DADO machine," Technical Report, Department of Computer Science, Columbia University, 1982. (Submitted to *ACM TOPLAS.*)

[27] Stolfo, S. J., "The DADO parallel computer," Technical Report Department of Computer Science, Columbia University, 1983. (Submitted to *AI Journal.*)

[28] Stolfo, S. J., "Five algorithms for PS execution on the DADO machine," Technical Report, Department of Computer Science, Columbia University, 1984. (Submitted to *AAAI 84.*)

[29] Stolfo, S. J., "On the design of parallel production system machines: What's in a LIP?," Technical Report, Department of Computer Science, Columbia University, 1984. (Submitted to *Int. Conf. on Fifth Generation Computer Systems.*)

[30] Stolfo, S. J., Vesonder, G. T., "ACE: An expert system supporting analysis and management decision making," *Bell System Technical Journal*, (To appear 1984).

[31] Taylor, S., C. Maio, S. J. Stolfo and D. E. Shaw, "PROLOG on the DADO machine: A parallel system for high-speed logic programming," *Proc. Third International Phoenix Conference on Computers and Communication*, 1984.