The DADO Production System Machine*

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

29 October 1984 CUCS-213-84

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 set of processing elements (PE's) (on the order of thousands), 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. Parallel algorithms for production system execution are briefly described. 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 calcuated from an analysis of simulations of the system.

*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, and IBM Corporations and the New York State Foundation for Advanced Technology. We gratefully acknowledge their support.

Table of Contents

1 Introduction 1
2 Expert Systems 1
2.1 Current Technology 1
2.2 Production Systems 2
2.3 Why a specialized production system architecture? 3
3 The DADO Machine 4
3.1 The System Architecture 4
3.2 The Binary Tree Topology 5
3.3 Production System execution 6
3.4 Algorithm 1: Full Distribution of PM 7
3.5 Algorithm 2: Original DADO Algorithm 8
3.6 Algorithm 3: Miranker's TREAT Algorithm 10
3.7 Algorithm 4: Fine-grain Rete 12
3.8 Algorithm 5: Multiple Asynchronous Execution 12
3.9 Comparison to other tree machines 16
4 The DADO Prototypes 17
4.1 Physical Characteristics 17
4.2 The Prototype Processing Element 18
4.3 The EPROM Resident Kernel 19
4.4 The DADO2 I/O Chip 21
5 Performance Evaluation of DADO2 23
5.1 Design Alternatives 23
5.2 Evaluation Method 23
5.3 Evaluation Results 23
6 Programming DADO 25
7 Conclusion 26

List of Figures

Figure 1:	Organization of a Problem-Solving Engine.	2
Figure 2:	An Example Production.	3
Figure 3:	Hyper-H embedding of a binary tree.	6
Figure 4:	The Leiserson chip design.	7
Figure 5:	The Leiserson printed circuit board.	8
Figure 6:	Abstract Production System Algorithm.	8
Figure 7:	Full Distribution of Production Memory.	9
Figure 8:	Functional Division of the DADO Tree.	10
Figure 9:	Original DADO Algorithm.	11
Figure 10:	The TREAT Algorithm.	13
Figure 11:	Fine-grain Rete Algorithm.	14
Figure 12:	Simple Multiple PS Program Execution.	15
Figure 13:	The DADO1 Prototype Processing Element.	19
Figure 14:	The DADO2 Prototype Processing Element.	20
Figure 15:	Illustration of Tree Neighbor Communication.	22
Figure 16:	Relative Performance of Four DADO PE Configurations.	24
Figure 17:	Performance Comparison of DADO1 and DADO2 of Similar Complexity on Variable Size Working Memory.	24

1 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 computer 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 [Buchanan and Feigenbaum 1978], medical diagnosis [Davis 1976], mineral exploration [Duda et al. 1979] and telephone cable maintenance [Stolfo and Vesonder 1982] 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.

Just as robitics 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 continue 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 expert systems. In the following sections we present an overview of DADO's application domain as well as the rationale for its design. Parallel algorithms for production system execution are then briefly described. We then detail the hardware design of the *DADO2* prototype, currently under construction at Columbia University, comprising *1023 microprocessors*. We conclude with a presentation of performance statistics recently calculated from simulations of the system. 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.

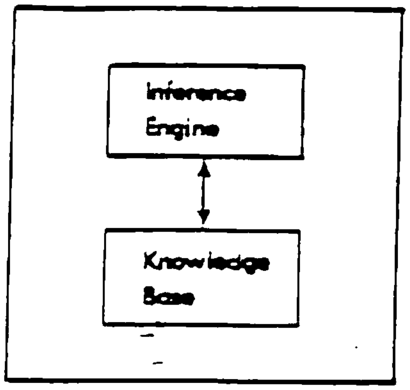
2 Expert Systems

2.1 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.

2.2 Production Systems

In general, a *Production System* [Newell 1973, Rychener 1976, Forgy and McDermott 1980] 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:

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 from WM
                and assert that =y is on top of the table.
```

----

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.

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.

2.3 Why a specialized production system architecture?

One problem facing expert systems technology is efficiency. It should be evident from the above description that large production system programs would spend most of their time in the match phase requiring an enormous number of primitive symbol manipulation tasks. 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 [McDermott 1981], 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 [Forgy 1982], 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 warant serious attention by parallel architects and VLSI designers.

Much of the experimental research conducted to date on specialized hardware for AI applications has focussed 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 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.

3 The DADO Machine

3.1 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 set of *processing elements* (PE's) (on the order of thousands), 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 [Flynn 1972]), the PE executes instructions broadcast by some ancestor PE within the tree. 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 [Siegel et al 1981] 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 PE version of DADO performs these operations in firmware embodied in its off-the-shelf components.

3.2 The Binary Tree Topology

As VLSI technology continues its downward trend in scaling, many PE's may be implemented on a single silicon chip. If the minimum feature size is halved, for example, four times as many components can be placed on a single chip. Thus, future microcomputer technology may provide additional speed, function and storage capacity of a single PE on a chip. Alternatively, as is the case with many of the approaches to fine-grain parallelism, many simpler processors may be integrated on the same chip. It is crucial, therefore, to interconnect a large number of processors in the most area-efficient topology possible. Further consideration must also be given to methods which efficiently drive the large number of device components to be placed on the chip, and which are not restricted by the severe pin-out limitations of packaging technology.

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 [Schwartz 1980] and binary tree machine architectures [Browning 1978]. Forgy [1980] proposed an interesting use of the mesh-connected ILLIAC IV machine [Lowrie et. al. 1975] 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 figure 3), 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 (see figure 4), first reported by Leiserson [1981], 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.

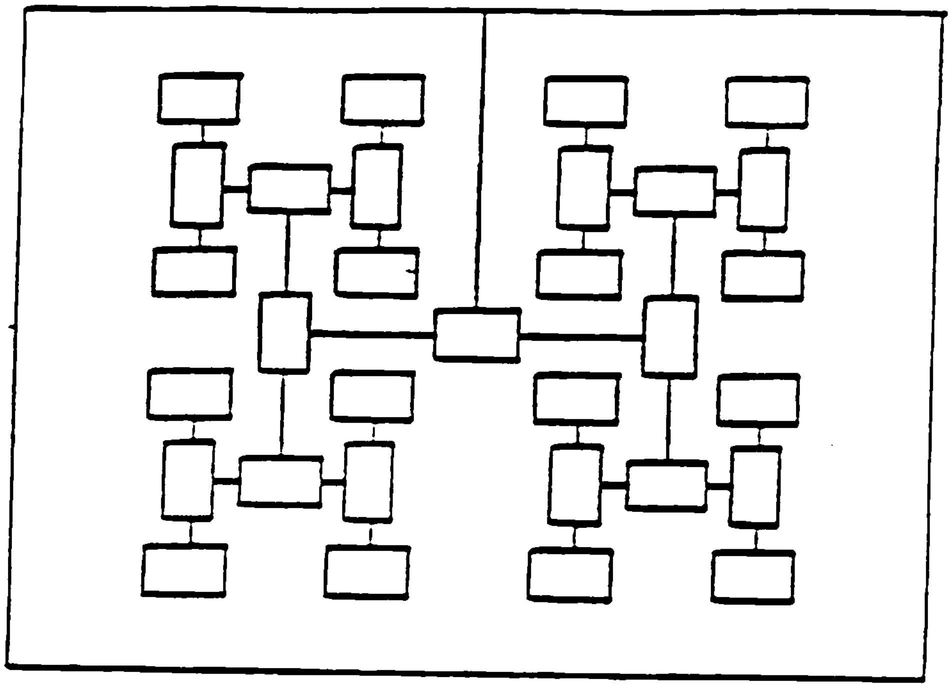
  - Pin-out on the Leiserson chip remains constant for any number of embedded PE's.

  - The Leiserson chip used with a simple recursive construction scheme produces printed circuit board designs (see figure 5) 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.

Figure 3: Hyper-H embedding of a binary tree.



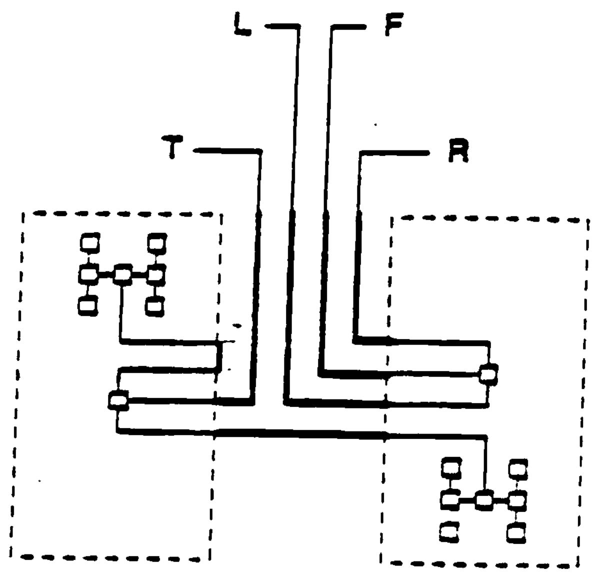
We note that binary trees do have certain limitations of practical importance. Although broadcasting a small amount of information to a large number of recipients is efficiently handled by binary trees, the converse is, in general, unfortunately not true. That is, for certain computational tasks (permutation 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.

3.3 Production System execution

In our earlier work, extensive theoretical analyses and software simulations of a high-speed algorithm for production system execution on DADO were completed and reported in [Stolfo and Shaw 1982]. Since that time we have invented a number of other related parallel algorithms. In this section we outline these five abstract algorithms. Each algorithm offers a number of advantages for particular types of production system programs. A more detailed treatment of these algorithms has appeared elsewhere [Stolfo 1984]. We expect to implement these algorithms on a DADO prototype and critically evaluate the performance of each on a variety of application programs. Software development is presently underway using a small DADO prototype that has been operational at Columbia University since April, 1983. We begin with a brief description of a general parallel approach to executing production systems.

On first glance it appears that each phase of the production system cycle is suitable for direct execution

Figure 4: The Leiserson chip design.



on parallel hardware, with the greatest opportunity for a speed-up in the match phase. This requires a partitioning of PM and WM among the available processors: some subset of processors would store and process the LHS of rules, while another possibly intersecting subset of processors would store and process WM elements. Thus, we envisage a set of processors concurrently executing pattern matching tests for a number of rules assigned to them. Similarly, once a conflict set of rules is formed, high-speed selection can be implemented in parallel as a logarithmic time algebraic operation. Finally, the RHS of a rule can be processed by a parallel update of WM. We summarize this approach by the abstract algorithm illustrated in figure 5.

This very simple view of the parallel implementation of the production system cycle forms the basis of our subsequent algorithms.

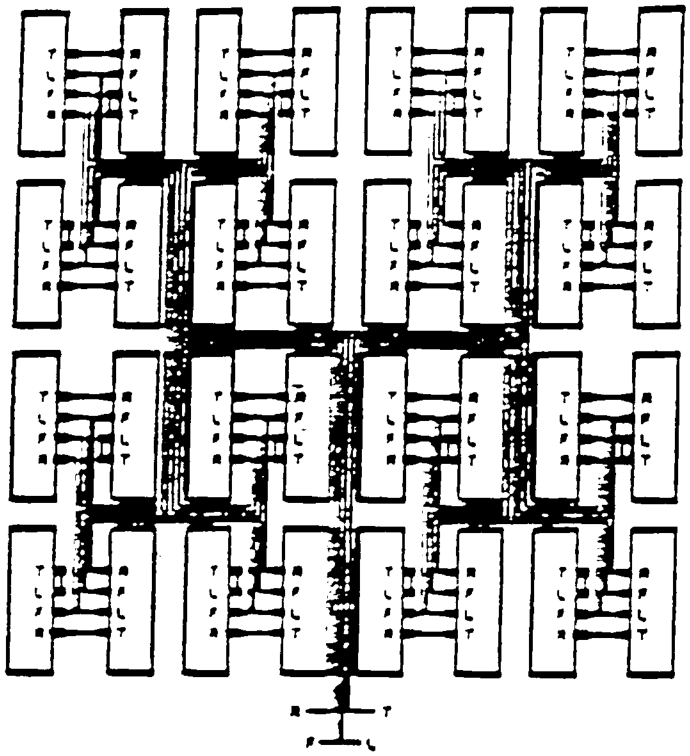
The reader is assumed to be knowledgeable about the Rete match algorithm (see [Forgy 1979] and [Forgy 1982]) for compiled production system programs. We will thus freely discuss the details of the Rete match when needed without prior explication.

3.4 Algorithm 1: Full Distribution of PM

In this case, a very small number of distinct production rules are distributed to each of the DADO PE's, as well as all WM elements relevant to the rules in question, i.e., only those data elements which match some pattern in the LHS of the rules. Algorithm 1 alternates the entire DADO tree between MIMD and SIMD modes of operation. The match phase is implemented as an MIMD process, whereas selection and act execute as SIMD operations.

In simplest terms, each PE executes the match phase for its own small production system. One such production system is allowed to "fire" a rule, however, which is communicated to all other PE's. The algorithm is illustrated in figure 6.

Figure 5: The Leiserson printed circuit board.



1. Assign some subset of rules to a set of (distinct) processors.

2. Assign some subset of WM elements to a set of processors (possibly distinct from those in step 1).

3. Repeat until no rule is active:

   a. Broadcast an instruction to all processors storing rules to begin the match phase, resulting in the formation of a local conflict set of matching instances.

   b. Considering each maximally rated instance within each processor, compute the maximally rated rule within the entire system. Report its instantiated RHS.

   c. Broadcast the changes to WM reported in step 3.b to all processors, which update their local WM accordingly. end Repeat;

Figure 6: Abstract Production System Algorithm.

3.5 Algorithm 2: Original DADO Algorithm

The original DADO algorithm detailed in [Stolfo 1983] makes direct use of the machine's ability to execute in both MIMD and SIMD modes of operation at the same point in time. The machine is logically divided into three conceptually distinct components: a *PM-level*, an *upper tree* and a number of *WM-subtrees* (see

1. Initialize: Distribute a simple rule matcher to each PE. Distribute a few distinct rules to each PE. Set CHANGES to initial WM elements.

2. Repeat the following:

3. Act: For each WM-change in CHANGES do:

    a. Broadcast WM-change (add or delete a specific WM element) to all PE's.

    b. Broadcast a command to locally match. [Each PE operates independently in MIMD mode and modifies its local WM. If this is a deletion, it checks its local conflict set and removes rule instances as appropriate. If this is an addition, it matches its set of rules and modifies its local conflict set accordingly].

    c. end do;

4. Find local maxima: Broadcast an instruction to each PE to rate its local matching instances according to some predefined criteria (conflict resolution strategy (see [McDermott and Forgy, 1978]).

5. Select: Using the high-speed max-RESOLVE circuit of DADO2, identify a single rule for execution from among all PE's with active rules.

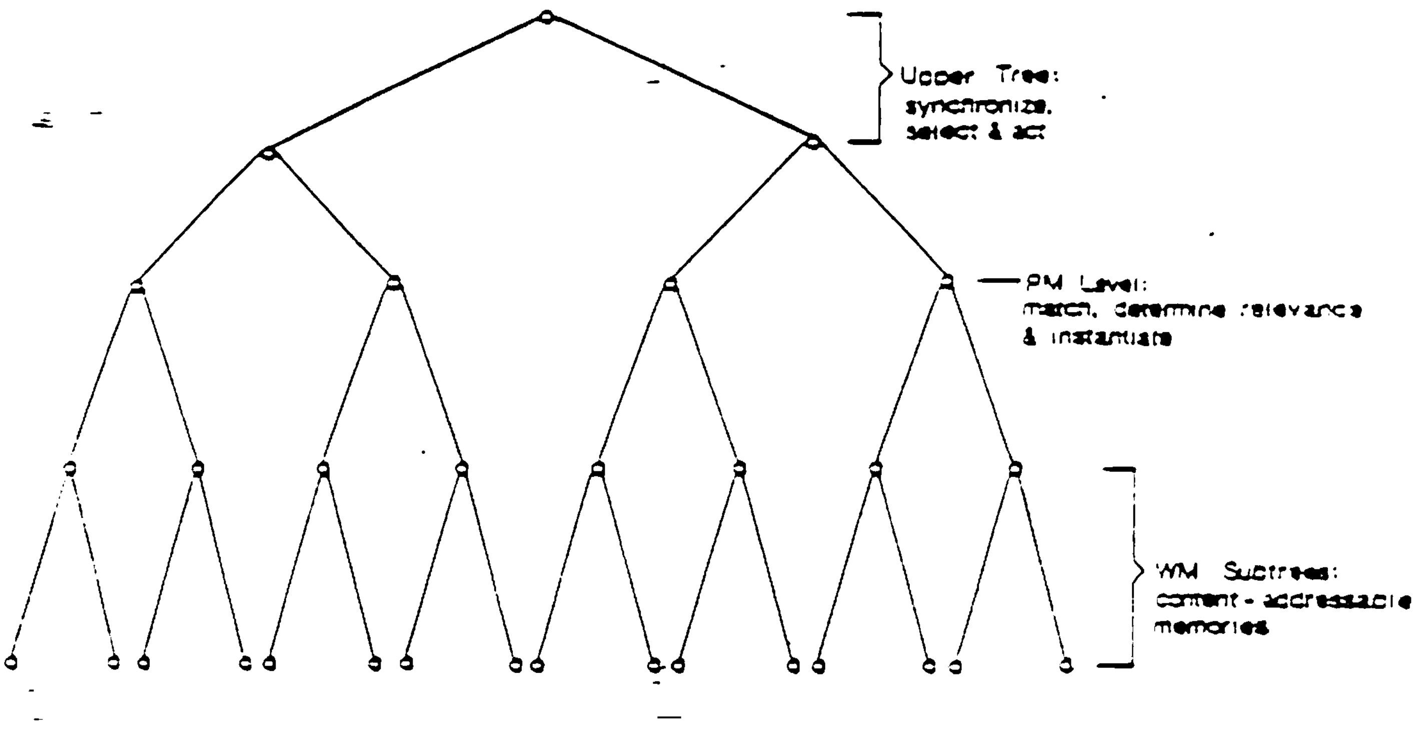
6. Instantiate: Report the instantiated RHS actions. Set CHANGES to the reported WM-changes.

7. end Repeat;

Figure 7: Full Distribution of Production Memory.

figure 8). The PM-level consists of MIMD-mode PE's executing the match phase at one appropriately chosen level of the tree. A number of distinct rules are stored in each PM-level PE. The WM-subtrees rooted by the PM-level PE's consist of a number of SIMD mode PE's collectively operating as a hardware content-addressable memory. WM elements relevant to the rules stored at the PM-level root PE are fully distributed throughout the WM-subtree. The upper tree consists of SIMD mode PE's lying above the PM-level, which implement synchronization and selection operations.

Figure 8: Functional Division of the DADO Tree.


Upper Tree:
synchronize, select & act
PM Level:
match, determine relevance & instantiate
WM Subtrees:
content - addressable memories

It is probably best to view WM as a distributed *relation*. Each WM-subtree PE thus stores relational tuples. The PM-level PE's match the LHS's of rules in a manner similar to processing relational queries. In terms of the Rete match, *intracondition* tests of pattern elements in the LHS of a rule are executed as relational *selection*, while *intercondition* tests correspond to *equi-join* operations. Each PM-level PE thus stores a set of relational tests compiled from the LHS of a rule set assigned to it. Concurrency is achieved between PM-level PE's as well as in accessing PE's of the WM-subtrees. The algorithm is illustrated in figure 8.

3.6 Algorithm 3: Miranker's TREAT Algorithm

Daniel Miranker has invented an algorithm which modifies Algorithm 2 to include several of the features of the Rete match for saving state. The *TRE*e *A*ssociative *T*emporally redundant (TREAT) algorithm [Miranker 1984] makes use of the same logical division of the DADO tree as in Algorithm 2. However, the state of the previous match operation is saved in distributed data structures within the WM-subtrees.

```
1. Initialize: Distribute a match routine and a partitioned subset of rules to each PM-level PE.
   Set CHANGES to the initial WM elements.

2. Repeat the following:

3. Act: For each WM-change in CHANGES do;

    a. Broadcast WM-change to the PM-level PE's and an instruction to match.

    b. The match phase is initiated in each PM-level PE:

        i. Each PM-level PE determines if WM-change is relevant to its local set of rules by
           a partial match routine. If so, its WM-subtree is updated accordingly. [If this is a
           deletion, an associative probe is performed on the element (relational selection) and
           any matching instances are deleted. If this is an addition, a free WM-subtree PE is
           identified, and the element is added.]

        ii. Each pattern element of the rules stored at a PM-level PE is broadcast to the
            WM-subtree below for matching. Any variable bindings that occur are reported
            sequentially to the PM-level PE for matching of subsequent pattern elements
            (relational equi-join).

        iii. A local conflict set of rules is formed and stored along with a priority rating in a
             distributed manner within the WM-subtree.

    c. end do;

4. Upon termination of the match operation, the PM-level PE's synchronize with the upper tree.

5. Select: The max-RESOLVE circuit is used to identify the maximally rated conflict set
   instance.

6. Report the instantiated RHS of the winning instance to the root of DADO.

7. Set CHANGES to the reported action specifications.

8. end Repeat;
```

Figure 9: Original DADO Algorithm.

TREAT views the pattern elements in the LHS of rules as relational algebra terms, as in Algorithm 2. Thus, the evaluation of such relational algebra tests is also executed within the WM-subtrees. State is saved in a WM-subtree in the form of distributed Rete *alpha memories* corresponding to partial selections of tuples matching various pattern elements. Rule instances in the conflict set computed on previous cycles are also stored in a distributed manner within the WM-subtrees. These two additions substantially improve the performance of Algorithm 2. (We note that Anoop Gupta of Carnegie-Mellon University independently analyzed a similar algorithm in [Gupta 1983]. Compared to Algorithm 2, TREAT should perform substantially better for temporally redundant systems. We note that Gupta's analysis of algorithm 2, however, depends on certain assumptions that derive misleading results.)

Another aspect of TREAT is the clever manner in which *relevancy* is computed. Pattern elements are first distributed to the WM subtrees. When a new WM element is added to the system, a simple match at each WM-subtree PE determines the set of rules at the PM-level which are affected by the change. Those identified rules are subsequently matched by the PM-level PE restricting the scope of the match to a smaller set of rules than would otherwise be possible with Algorithm 2.

The TREAT algorithm is outlined in figure 9.

3.7 Algorithm 4: Fine-grain Rete

A Rete network compiled from the LHS's of a rule set consists of a number of simple nodes encoding match operations. Tokens, representing WM modifications, flow through the network in one direction and are processed by each node lying on their traversed paths. Fortunately, the maximum fan-in of any node in a Rete network is two. Hence, a Rete network can be represented as a binary tree (with some minimal amount of node splitting).

This observation leads to Algorithm 4 whereby a logical Rete network is embedded on the physical DADO binary tree structure. In the simplest case, leaf nodes of the DADO tree store and execute the initial linear chains of one-input test nodes, whereas internal DADO PE's execute two-input node operations. The physical connections between processors correspond to the logical data flow links in the Rete network. The entire DADO machine operates in MIMD mode while executing this algorithm, behaving much like a pipelined data flow architecture.

Algorithm 4 is illustrated in figure 10.

3.8 Algorithm 5: Multiple Asynchronous Execution

In our discussion so far, no mention was made about multiple rule firings. We may view this as

- multiple, independently executing production system programs, or

- executing multiple conflict set rules of the same proudction system program concurrently.

In this regard we offer not a single algorithm, but rather an observation that may be put to practical use in each of the abovementioned algorithms.

We note that any DADO PE may be viewed as a root of a DADO machine. Thus, any algorithm operating at the physical root of DADO may also be executed by some descendant node. Hence, any of the aforementioned algorithms can be executed at various sites in the machine concurrently! (This was noted in [Stolfo and Shaw 1982].) This coarse level of parallelism, however, will need to be controlled by some algorithmic process executed in the upper part of the tree. The *simplest* case is represented by the procedure illustrated in figure 11, which is similar in some respects to Algorithm 2.

In the cases where various PS-level PE's need to communicate results with eachother, step 3 is replaced

```
1. Initialize: Distribute to each PM-level PE a simple matcher (described below) and a compiled
   set of rules. Distribute to the WM-subtree PE's the appropriate pattern elements appearing in
   the LHS of the rules appearing in the root PM-level PE. Set CHANGES to the initial WM
   elements.

2. Repeat the following:

3. Act: For each WM-change in CHANGES do;

    a. Broadcast WM-change to the WM-subtree PE's.

    b. If this change is a deletion, broadcast an instruction to match and delete WM elements
       and any affected conflict set instances calculated on previous cycles.

    c. Broadcast an instruction to PM-level PE to enter the Match Phase.

    d. At each PM-level PE do;

        i. Broadcast to WM-subtree PE's an instruction to match the WM-change against
           the local pattern element.

        ii. Report the affected rules and store in L.

        iii. Order the pattern elements of the rules in L appropriately.

        iv. For each rule in L do;

            1. Match remaining patterns of the rules specified in L as in Algorithm 2.

            2. For each new instance found, store in WM-subtree with a priority rating.

            3. end do;

        v. end do;

    e. end for each;

4. Select: Use max-RESOLVE to find the maximally rated instance in the tree.

5. Report the winning instance.

6. Set CHANGES to the instantiated RHS of the winning rule instance.

7. end Repeat;
```

Figure 10: The TREAT Algorithm.

```
1. Initialize: Map and load the compiled Rete network on the DADO tree. Each node is provided
   with the appropriate match code and network information (see [Forgy 1982] for details). Set
   CHANGES to initial WM elements.

2. Repeat the following:

3. Act: For each WM-change in CHANGES do;

      a. Broadcast WM-change (a Rete token) to the DADO leaf PE's.

      b. Broadcast an instruction to all PE's to Match. (First, the leaf processors execute their
         one-input test sequences on the new token. The interior nodes lay idle waiting for match
         results computed by their descendants. Those tokens passing the one-input tests are
         communicated to the immediate ancestors which immediately begin processing their two-
         input tests. The process is then repeated until the physical root of DADO reports
         changes to the conflict set maintained in the DADO control processor).

      c. end do;

   Select: The root PE is provided with the chosen instance from the control processor. Set
   CHANGES to the instantiated RHS.

4. end Repeat;
```

Figure 11: Fine-grain Rete Algorithm.

```
1. Initialize. Logically divide DADO to incorporate a static Production System-Level (PS-level),
   similar to the PM-level of Algorithm 2. Distribute the appropriate production system program
   to each of the PE's at the PS-level.

2. Broadcast an instruction to each PS-level PE to begin execution in MIMD mode. (Upon
   completion of their respective programs, each PS-level PE reconnects to the tree above in
   SIMD mode.)

3. Repeat the following.

     a. Test if all PS-level PE's are in SIMD mode.

   End Repeat;

4. Execution Complete. Halt.
```

Figure 12: Simple Multiple PS Program Execution.

with appropriate code sequences to report and broadcast values from the PS-level in the proper manner. Each of the programs executed by PS-level PE's are first modified to synchronize as necessary with the root PE to coordinate the communication acts, at, for example, termination of the Act phase.

In addition to concurrent execution of multiple production system programs, methods may be employed to concurrently execute portions of a single production system program. These methods are intimately tied to the way rules are partitioned in the tree. Subsets of rules may be constructed by a static analysis of PM separating those rules which do not directly interact with each other. In terms of the *match* problem-solving paradigm, for example, it may be convenient to think of independent subproblems and the *methods* implementing their solution (see [Newell 1973]). Each such method may be viewed as a high-level subroutine represented as an independent rule set rooted by some internal node of DADO. Algorithm 1, for example, may be applied in parallel for each rule set in question. Asynchronous execution of these subroutines proceeds in a straight forward manner. The complexity arises when one subset of rules infers data required by other rule sets. The coordination of these communication acts is the focus of our ongoing research. Space does not permit a complete specification of this approach, and thus the reader is encouraged to see [Ishida 1984] for details of our initial thinking in this direction.

Of the five reported algorithms, only the original DADO algorithm (number 2) has been carefully studied analytically. The performance statistics of the remaining four algorithms have yet to be analyzed in detail. However, much of the performance statistics cannot be analyzed without specific examples and detailed implementations. In the course of the next year of our research we intend to implement each of the stated algorithms on a working prototype of DADO.

Although analytical studies and software implementations are primary tasks of the DADO project, our current efforts have focussed 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 larger 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.

3.0 Comparison to other tree machines

It should be noted that many of the decisions made in designing DADO were influenced by the organization of the NON-VON1 supercomputer [Shaw 1982] and the Caltech tree machines [Browning 1980]. Perhaps the best way to distinguish DADO from these two tree machine architectures is by considering the modes of execution of each of the constituent PE's, and the implications for the hardware design.

The proposed Caltech tree machine is a full MIMD device incorporating thousands of PE's in a full-scale version. Each PE executes its own independent program and thus requires a substantial amount of local memory as is the case in the DADO machine. Communication is supported by a buffered message passing protocol, where the recepient of each message is identified by relatively complex I/O circuitry at each node. Other forms of communication (for example, global broadcast) are implemented by sequential logic.

NON-VON1, by comparison, is a full SIMD, massively-parallel synchronous device incorporating millions of simple, *highly-area efficient* PE's, each associated with only 64 bytes of local RAM. In general, each NON-VON1 PE executes an instruction broadcast from a single control processor, located at the root of the tree, and thus requires a highly-efficient method of global broadcast. The I/O switch design incorporated within each node of the NON-VON1 tree contains a few inverters driving the signals along the broadcast bus, and therefore communication is implemented by high-speed combinational logic.

DADO, on the other hand, is capable of executing in both SIMD and MIMD modes, and thus contains elements of both machine designs. DADO incorporates a combinational I/O switch similar to that employed in NON-VON. However, each DADO PE may drive the I/O switch, in addition to the single coprocessor of DADO. Thus, DADO also supports very high speed global broadcast. However, because of the replication of substantial programs within various PE's in the tree, a DADO PE has been designed with a more general (8 bit) processor as well as an 16K byte RAM. Thus, DADO cannot achieve the same processor density as is possible in NON-VON.

The DADO design attempts to synergistically merge the advantages of both the NON-VON and the Caltech tree machine. Indeed, DADO can be easily programmed to simulate both proposed designs. It is not clear whether or not the NON-VON approach to single-instruction stream, massive parallelism will be substantially limited by its inability to execute independent coarser-grain programs concurrently. For production system programs, that appears to be the case. Nor is it clear whether or not the Caltech approach of large-scale parallelism, albeit substantially lower than that of NON-VON for certain computational problems, can achieve the same throughput projected for NON-VON. It is our hope that experimentation with the DADO prototype may provide some of these answers, and begin to elucidate the precise nature of the tradeoffs involved with both approaches.

4 The DADO Prototypes

4.1 Physical Characteristics

As noted, 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 magahertz producing 4 million instructions per second (MIPS) (see [Miranker 1984b]). (The effective useable 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. 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 a Hewlett-Packard 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.

4.2 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. 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, an 8K 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 eraseable programmable read only memory (EPROM), and a 256-byte RAM on a single silicon chip. The incorporation of an EPROM in each DADO2 PE provides a suitable measure of conservative safeguarding if we ever encounter bugs that need to be repaired when the prototype is fully configured. 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. DADO1 was implemented within 4 months of delivery of the hardware components. Figure 13 illustrates the DADO1 prototype PE while figure 14 illustrates DADO2's PE.

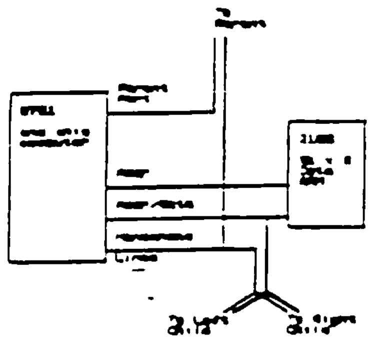
Note that the same processor connections exist in the DADO2 PE design as those appearing in the DADO1 design. If in the unlikely event that the planned I/O chip does not function properly, DADO2 will thus remain operational, but will not run as fast as envisaged. Since the DADO1 hardware to date has 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 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 instructions per second would be achieved on DADO2 without the I/O chip, as detailed in a 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

Figure 13: The DADO1 Prototype Processing Element.



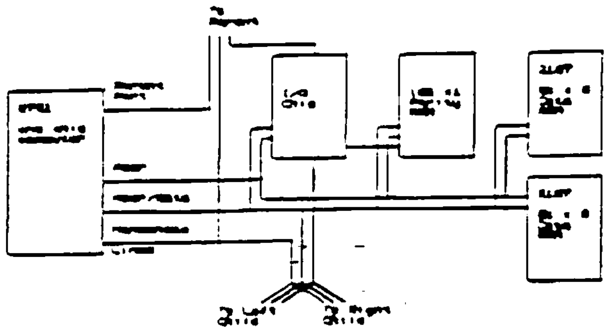
the realm of feasibility to implement a DADO2 PE on a single silicon chip. Thus, although DADO2 may appear impressive (an inexpensive, compact system with a thousand computers executing roughly 600 million insructions per second) its design is very conservative and probably at least an order of magnitude 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.)

4.3 The EPROM Resident Kernel

Each 8751 processor in the DADO machine contains an identical program stored within its EPROM. This program, called the DADO kernel, implements the execution semantics of a DADO PE, (the MIMD and SIMD modes of operation), and is in a sense the microcode of the DADO machine. The kernel may also be viewed as an operating system. It contains low level I/O drivers, code for higher level DADO communication primitives and run time support procedures. PPL/M, detailed in a subsequent section, is the high level parallel language designed for system-level programming of DADO.

The DADO kernel and the PPL/M compiler are tightly integrated. The advantages of restricting access to the machine through a high level language are numerous. Most importantly, the productivity of the DADO system programmers is enhanced by programming only in a high level language as opposed to assembly language. Further, the parallel programming constructs are clearly defined and the DADO kernel need only support a small number of well defined primitives. Lastly, PPL/M is strongly typed and the compiler prevents the system programmer from directly accessing the kernel. Therefore, the kernel can be made robust without extensive error and parameter checking.

Figure 14: The DADO2 Prototype Processing Element.



The kernel has two top level interpreters, one for MIMD mode behavior of a PE and one for SIMD mode. Monitor calls made by run time software may change the state of a PE from one mode to the other. In MIMD mode, code is conventionally fetched and executed from the PE's local RAM. SIMD instruction blocks may be embedded within such code. When a SIMD block is encountered a monitor call is made with a pointer to the SIMD block. The kernel then broadcasts the instruction stream to the descendant PE's and subsequently executes the instructions directly.

The SIMD mode interpreter executes a simple loop: read the instructions broadcast from its parent, pass them to its children and if the PE is in SIMD enabled state execute them. If the SIMD instruction is a DADO communication primitive, control is passed to an appropriate monitor routine.

The DADO communication primitives are implemented with six low level I/O functions. Each function performs a read or write operation with each of a PE's three tree neighbors. For example, the DADO primitive "SEND(RC)" simultaneously moves a byte from each PE's variable called A8, to a second variable IO8, in the PE's right child (see figure 15). This operation is illustrated by the following code sequence. The distinguished "enable" variable, EN1, appearing in each PE is set to the current SIMD mode state of the processor: either **enabled** or **disabled**.

```
SEND_RC: procedure ();
     declare temp byte;       /* Local temporary.  */

     temp= Read_P;            /* Read byte from parent. */
     Write_RC(A8);            /* Write byte to left child.*/
     if EN1 then IO8 = temp; /* If enabled, accept the byte. */
end;
```

The low level routines "Write_RC" and "Read_P" act as no-ops if the communication is performed with a logically disconnected neighboring PE. The PPL/M compiler insures that this kernel routine is called within every SIMD PE at the same time. The low level I/O functions execute the primitive steps of a four cycle handshake protocol which forces the appropriate synchronization of the operations.

A complex issue arises when defining communication among a group of processors, some of which may be in SIMD disabled state. What should a SIMD disabled processor do with a byte received from an enabled processor? Similarly, what should happen if the communication primitive directs a disabled processor to send a byte to an enabled one? These issues have been resolved using the following convention:

> A SIMD processor executes all instructions whether it is enabled or disabled. However, instructions executed by a SIMD disabled processor have no local side effects.

In effect a disabled processor may communicate data to a neighbor, but data received by the disabled processor is ignored. Thus, the behavior of a PE is determined only by information local to the PE. In the above code segment for example, the kernel routine completely performs the I/O operation and then explicitly tests the enable flag before introducing the side effect.

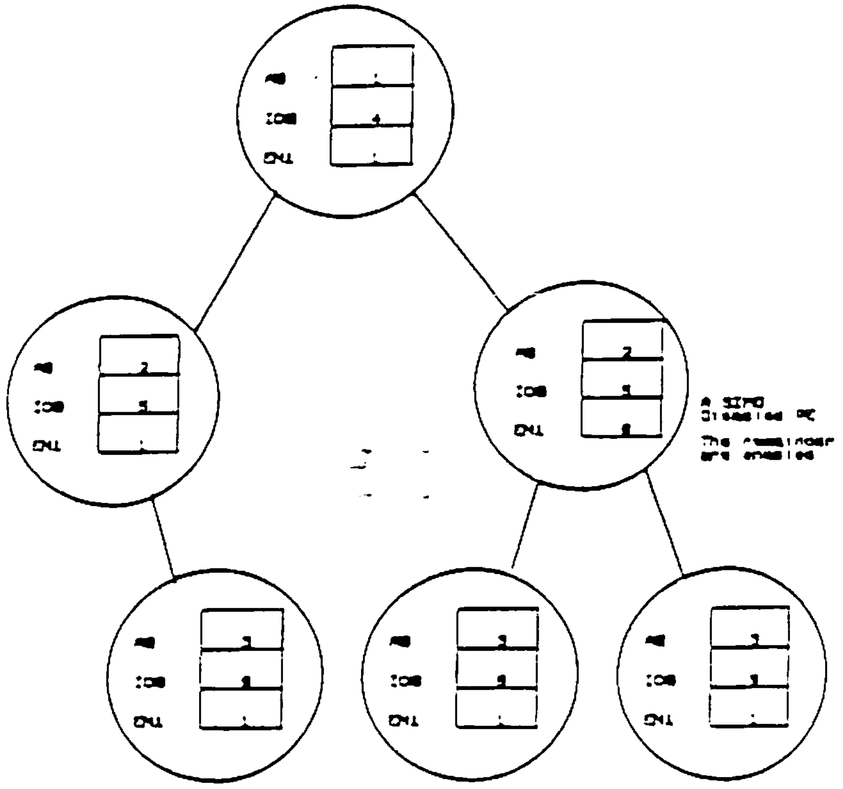
4.4 The DADO2 I/O Chip

In the DADO1 prototype all communication operations are performed directly by firmware in the processor's kernel. For each byte moved along a tree edge 12 instruction cycles are consumed to execute a four cycle handshake. Though computationally expensive, this implementation is quite expedient for the rapid prototyping of a small machine. However, a 1023 node DADO2 is 10 levels deep. Thus, if DADO2 were implemented using the same strategy an instruction broadcast from the root of the tree to a leaf would require at least 120 instruction cycles. For the larger machine we decided to construct a small circuit to improve global communication: Broadcast, Report, and Resolve (discussed below). Despite using a synchronous bit-serial protocol, the current I/O chip design is able to broadcast or report a byte value thoughout the machine in less than one 8751 instruction cycle.

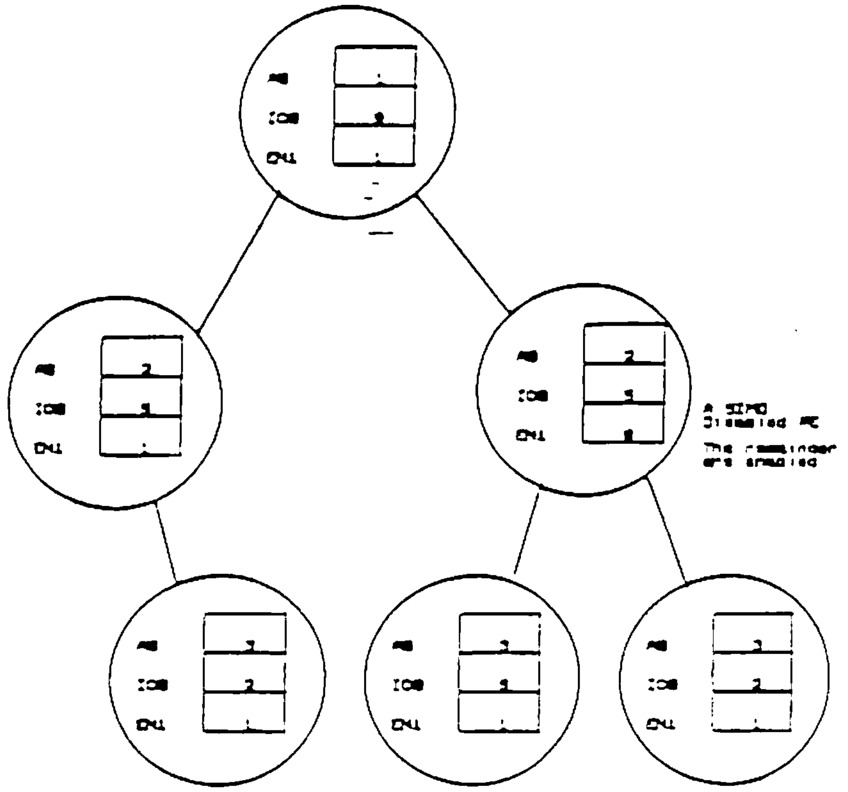
We have also included in the design memory support logic with parity checking as well as a global interrupt mechanism. The global interrupt mechanism permits any PE in the machine to initiate an interrupt in every PE and the host coprocessor. After an interrupt the machine may perform a context switch. Currently under development is a symbolic debugging program for a parallel programming environment which uses this hardware capability.

The I/O chip is being implemented in a small, 1000 gate, semi-custom gate array chip. The logic design and simulation is being done with the aid of a Valid Logic Systems Scald design station, a state of the art logic design system. The semi-custom I/O chips will be fabricated under contract with LSI Logic Corporation.

Figure 15: Illustration of Tree Neighbor Communication.



State of a SIMD subtree before a Send(RC) instruction



State of a SIMD subtree after a Send(RC) instruction

5 Performance Evaluation of DADO2

5.1 Design Alternatives

As noted, 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 handle global communication among the PE's 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 8751 instruction cycle. This efficiency gain does not come free. The I/O circuit increases a PE's component count. The appropriateness of the I/O circuit was determined by investigating the relative performance of a machine incorporating the I/O circuit, and a machine without the I/O circuit and using the remaining board 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 to the SIMD PE's. In a typical SIMD machine a control processor issues a stream of machine level instructions that are executed synchronously in lock step by all 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, an instruction should be made 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 match 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 additional overhead which may decrease the overall performance of the system.

5.2 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). 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, [Sauer 1982]. 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 [Miranker 1983].

5.3 Evaluation Results

Figure 16 summarizes the relative throughput of the four configurations working on a problem typical of the size we expect a 1023 node DADO to handle: 1-2000 productions and 1000 working memory elements. 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 16: Relative Performance of Four DADO PE Configurations.

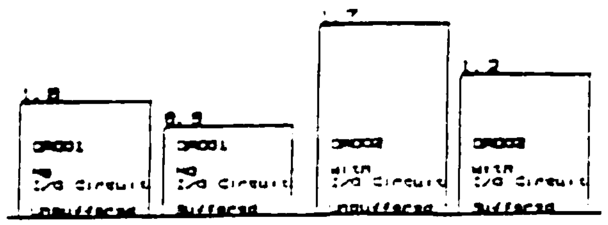
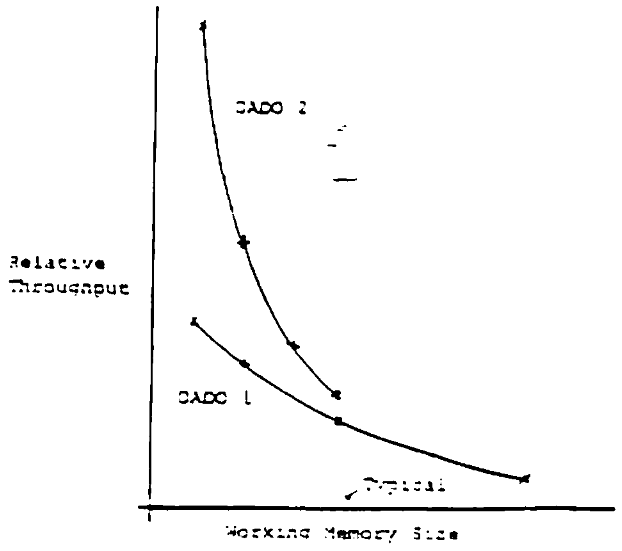


Figure 17 is a comparison of a 5 level DADO subtree (consisting of 31 PE's) without the I/O circuit, and a 4 level DADO subtree, (consisting of 15 PE's) with the I/O circuit. The x-axis represents a rough approximation of the number of WM elements in the system. The graph shows that for typical size problems 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.

Figure 17: Performance Comparison of DADO1 and DADO2 of Similar Complexity on Variable Size Working Memory.



6 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 (AI, IO8 and EN1, cited above, for example). 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 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-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.

As noted, 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 as in DADO1. 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.

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 arbitrarily 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 descendent 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 by the kernel by propagating 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 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.

7 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 forward-chaining 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 [Taylor et al. 1983].

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 [Griss 1981] for this purpose due to its relative ease in porting to a new processor. In [Weisberg et. al. 1984] we report on the current status of the ||PSL (parallel PSL) implementation and note its relationship to PPL/M.

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 *55 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 this larger 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 based on Intel 8751 technology achieves a 50% performance improvement over a machine roughly six times its size. Lastly, had we used a processor technology based on the Motoroila 68000, for example, DADO2 would achieve an execution rate of over 1300 production system cycles per second! DADO3 should perform substantially better.

Acknowledgements

It is a great pleasure to thank the many people and organizations who have contributed to the DADO project. Daniel Miranker, a Ph.D. student at Columbia, is responsible for many of the detailed hardware and software designs of the machine and is the driving force in the effort to implement the OPS family of languages on the system. Stephen Taylor, also a Ph.D. student, working closely with Chris Maio, Andy Lowry and faculty co-investigator Professor Gerald Maguire have made tremendous progress in specifying the execution of PROLOG on DADO. Their design of the PROLOG system is as elegant as the hardware solutions provided by our project engineer, Shunsaku Ueda, who deserves special mention. Many researchers have contributed in substantial ways, too numerous to specify in great detail. We thus would like to acknowledge the efforts of Eugene Dong, Andrew Comas, Mark Lerner, Jody Weiss, Mike Weisberg, Jim Gilpatrick, Janvid Cheng, Wai Man Wong, Monique Fei, Daphne Tzoar, Doug Degroot, Alex Pasik and Ted Sabety. We would also like to thank Lanny Forgy, Allen Newell and Mike Rychener for very interesting and thought-provoking conversations about DADO.

The Defense Advanced Research Projects Agency is our primary source of support through contract N00039-82-C-0427. Intel Corporation has contributed most of the components used in the construction of DADO1, and continues to support our development effort for DADO2 by providing additional hardware components. Digital Equipment Corporation and Hewlett-Packard Corporation have provided computational resources for our software development. Valid Logic Systems has donated the prototype boards used in the construction of DADO1 and has continued to aid our research. Recently the DADO project has received support from the New York State Foundation for Advanced Technology, for which we are grateful. Finally, we acknowledge the assistance of IBM Corporation for providing components for DADO2 as well as taking a more active role in supporting our research.

REFERENCES

Browning, S., "Hierarchically Organized Machines", In Mead and Conway (Eds.), *Introduction to VLSI Systems*, 1978.

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

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

Davis, R. and J. King. *An Overview of Production Systems*. Technical Report, Stanford University Computer Science Department, 1975.

Davis, R. *Applications of meta-level knowledge to the construction, maintenance and use of large knowledge bases*, Rep. No. STAN-CS-76-552, Computer Science Department, Stanford University, 1976.

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, 153-167, 1979.

Flynn, M. J., "Some Computer Organizations and Their Effectiveness", *IEEE Transactions on Computers*, 1972.

Forgy, C. L., *A Note on Production Systems and ILLIAC IV*, Technical Report 130, Department of Computer Science, Carnegie-Mellon University, 1980.

Forgy, C. L., Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem, *Artificial Intelligence 19*, 1982

Griss, M.L. and A. C. Hearn, A Portable LISP Compiler, *Software-Practice and Experience*, 11, 1981.

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

Ishida, T., "Asynchronous Parallel Execution of Production Systems on a Tree-structured Machine", Technical Report, Department of Computer Science, Columbia University, 1984.

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

University, 1981.

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

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

Miranker, D. P., "Performance Analysis of Two Competing DADO PE Designs", Technical Report, Department of Computer Science, Columbia University, 1983.

Miranker, D. P., "The System-level design of the DADO1 Prototype", Technical Report, Department of Computer Science, Columbia University, 1984. (in preparation)

Miranker, D. P., "Production System Execution on DADO ", Technical Report, Department of Computer Science, Columbia University, 1984. (in preparation)

Newell, A., "Production Systems: Models of Control Structures", In W. Chase (editor), *Visual Information Processing*, Academic Press, 1973.

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

Sauer, C. H., E. A. Macnair, and J. F. Kurose, *The Research Queueing Package, CMS User's Guide.* Technical Report RA 139 #41127, IBM Research Division, 1982

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

Shaw, D. E., *The NON-VON Supercomputer*,Technical Report, Department of Computer Science, Columbia University, 1982.

Siegel, H. J., L. J. Siegel, F. C. Kemmerer, P. T. Mueller, H. E. Smolky and D. S. Smith, PASM: A Parititionable SIMD/MIMD System for Image Processing and Pattern Recognition, *IEEE Tran. on Computers*, 1981.

Stolfo, S. J. and D. E. Shaw. *Specialized Hardware for Production Systems*. Technical Report, Department of Computer Science, Columbia University, 1981.

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.

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., D. Miranker and D. E. Shaw, "Architecture and Applications of DADO: A Large-scale Parallel Computer for Artificial Intelligence", in *Proceedings International Joint Conference on Artificial Intelligence*, Karslruhe, West Germany, 1983.

Stolfo, S. J., "The DADO Parallel Computer", Technical Report, Department of Computer Science, Columbia University, 1983.

Stolfo, S. J., and G. T. Vesonder, ACE: An Expert System Supporting Analysis and Management Decision Making, 1982. (to appear in the *Bell System Technical Journal*.)

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.

Taylor, S., S.G. Maguire, S. Stolfo and A. Lowry, "Implementing PROLOG using parallel associative operations", *Proc. Logic Programming Conference*, Atlantic City, 1984.

Weisberg, M., M. Lerner, G. Maguire and S. J. Stolfo, "||PSL: A Parallel Lisp for Programming the DADO Machine", Technical Report, Department of Computer Science, Columbia University, 1984.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE. DISTRIBUTION UNLIMITED.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION COLUMBIA UNIVERSITY	6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION NAVELEX
6c. ADDRESS *(City, State, and ZIP Code)* 450 Computer Science Building Columbia University New York, NY 10027		7b. ADDRESS *(City, State, and ZIP Code)* 2511 Jefferson Davis Highway Arlington, VA 22202
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA	8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*
1400 Wilson Boulevard
Arlington, VA 22209

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO
	N00039-84-C-0165	2	

11. TITLE *(Include Security Classification)*
The DADO Production System Machine

12. PERSONAL AUTHOR(S)
Stolfo, S. J. and D. P. Miranker

13a. TYPE OF REPORT SPECIAL	13b. TIME COVERED FROM 6/84 TO 9/84	14. DATE OF REPORT *(Year, Month, Day)* 1984, October 29	15. PAGE COUNT 30

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	DADO, Production Systems, Parallel Computer, Fifth Generation, Logic Programming, AI, LISP.

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

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 set of processing elements (PE's) (on the order of thousands), 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. Parallel algorithms for production system execution are briefly described. 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 calcuated from an analysis of simulations of the system.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Salvatore J. Stolfo	22b. TELEPHONE *(Include Area Code)* (212) 280-3111	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE