DADO: A TREE-STRUCTURED MACHINE ARCHITECTURE FOR PRODUCTION SYSTEMS*

Salvatore J. Stolfo
and
David Elliot Shaw

Columbia University

ABSTRACT

DADO is a parallel tree-structured machine designed to provide highly significant performance improvements in the execution of large Production Systems. The DADO machine comprises a large (on the order of a hundred thousand) set of processing elements (PE's), each containing its own processor, a small amount (2K bytes, in the current 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 a general procedure for the parallel execution of production systems on the DADO machine, and outlines in general terms how this procedure can be extended to include commutative and multiple, independent production systems.

1. Introduction

DADO [Stolfo and Shaw, 1981] is a parallel, tree-structured machine designed to provide highly significant performance improvements in the execution of production systems. A production system [Newell, 1973; Davis and King 1975; Rychener, 1976] 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 which is to be added to (asserted) or removed from WM when the LHS successfully matches against the contents of WM.

In operation, the PS repeatedly executes the following cycle of operations:

1. Match: For each rule, determine whether the LHS matches the current environment of WM.
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.

In this paper, data elements in WM will have the form of arbitrary ground literals in the first order predicate calculus. For pedagogical reasons, we will restrict our attention to the case in which both the LHS and RHS are conjunctions of predicates in which all first order terms are composed of constants and existentially quantified variables. (DADO in fact supports the incorporation of universally quantified variables in the LHS of a production as well, but an adequate treatment of this case would substantially complicate our exposition, and has thus been omitted. The interested reader is referred to a discussion of the LSEC algorithm for logical satisfaction, presented in a doctoral dissertation by Shaw [1980].) A negated pattern in the LHS causes the matching procedure to fail whenever WM contains a matching ground literal, while a negated pattern in the RHS causes all matching data elements in WM to be deleted.

An example production is presented in Figure 1-1. (Variables are prefixed with an equal sign.)

```
(Part-category =part electronic-component)
(Used-in =part =product)
(Supplied-to =product =customer)
(NOT Manufactured-by =part =customer)
  --> (Dependent-on =customer =part)
      (NOT Independent =customer)
```

Figure 1-1: An Example Production

Because the matching of each rule against WM is essentially independent of the others (at least in the absence of contention for data in WM), it is natural to attempt a decomposition of the matching portion of each cycle into a large number of tasks suitable for physically concurrent execution on parallel hardware. While the design of special-purpose parallel machines adapted to artificial intelligence applications has attracted some attention [Fahlman, 1979; Fuhlrott, 1982], little

*This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-82-C-0427.

progress has been made in the application of highly concurrent hardware to the execution of rule-based systems. Forgy [1980] proposed a very interesting use of the ILLIAC IV machine for such applications, but recognized that his approach failed to identify all matching productions under certain circumstances.

In this paper, we describe a tree-structured machine architecture that utilizes the emerging technology of VLSI systems in support of the highly efficient parallel execution of large-scale production systems. Portions of the machine, which we have come to call DADO, are now in the early stages of construction at Columbia University. We believe a full-scale DADO prototype, capable of significant performance improvements over implementations based on von Neumann machines, to be technically and economically feasible for implementation using current technology.

2. The DADO Machine Architecture

The DADO machine comprises a very large (on the order of a hundred thousand) set of processing elements (PE's), each containing its own processor, a small amount (2K bytes, in the current design) of local random access memory, and a specialized I/O switch. The PE's are interconnected to form a complete binary tree. Certain aspects of the DADO machine are modelled after NON-VON [Shaw, 1979; Shaw, et al., 1981], a tree-structured, highly parallel machine containing a larger number of much simpler processing elements.

In NON-VON, most of the PE's are severely restricted in both processing power and storage capacity, and are thus not typically used to execute independent programs. Instead, a single control processor, located at the root of the NON-VON tree, typically broadcasts a single stream of instructions to all PE's in the tree. Each such instruction is then simultaneously executed (on different data) by every PE in the tree. This mode of operation has been referred to in the literature of parallel computation as single instruction stream, multiple data stream (SIMD) execution [Flynn, 1972]. (The above description is in fact somewhat oversimplified, since NON-VON in fact permits independent instruction streams to be broadcast to selected subtrees. Such subtrees, though, must be rooted at a single, fixed level within the tree, where additional processing power is available.)

Within the DADO machine, on the other hand, each PE is capable of executing in either of two modes. In the first, which we will call SIMD mode, the PE executes instructions broadcast by some ancestor PE within the tree, as in the NON-VON machine. 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.

When a DADO PE enters MIMD mode, its I/O switch settings are 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, NON-VON-like SIMD device in which all PE's execute a single instruction at a given point in time.

As in NON-VON, the DADO I/O switch supports communication between physically adjacent neighbors (parents and children) within the tree in addition to broadcast-based communication.

3. Allocation of Productions and Working Memory

In order to execute the production system cycle, the I/O switches are configured in such a way as to divide the DADO machine into three conceptually distinct components. One of these components consists of all PE's at a particular level within the tree, called the PM level, which is chosen in a manner to be detailed shortly. The other two components are the upper portion of the tree, which comprises all PE's located above the PM level, and the lower portion of the tree, which consists of all PE's found below the PM level. This functional division is illustrated in Figure 3-1.

Each PE at the PM level is used to store a single production. The PM level must thus be chosen such that the number of nodes at that level is at least as large as the number of productions in PM. The subtree rooted by a given PE at the PM level will store that portion of WM that is relevant to the production stored in that PE. A ground literal in WM is defined to be relevant to a given production if its predicate symbol agrees with the predicate symbol in one of the pattern literals in the LHS of the production, and all constants in the pattern literal are equal to the corresponding constants in the ground literal. Intuitively, the set of ground literals relevant to a given production consists of exactly those literals that might match that production, given appropriate variable bindings.

The constituent subtrees that make up the lower portion of the tree will be referred to as the WM-subtrees. For simplicity, we will assume in this paper that each PE in a WM-subtree rooted by some production contains exactly one ground literal relevant to that production. (Using "packing" techniques analogous to those employed in NON-VON, however, this assumption is easily relaxed at the expense of a modest cost in time.) It should be noted that, since a single ground literal may be

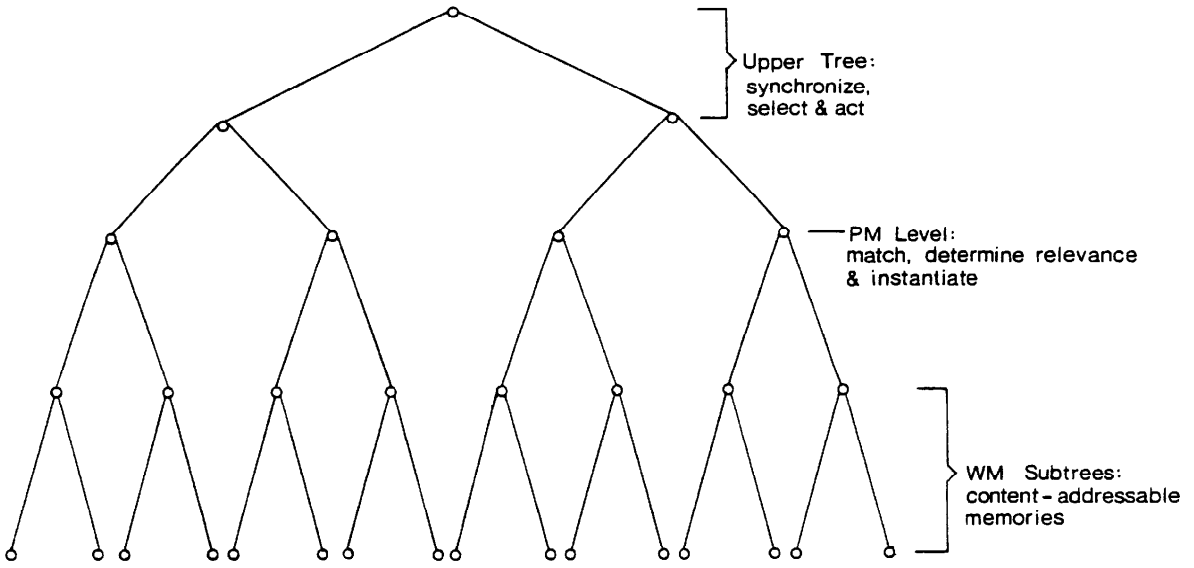

Figure 3-1: Functional Division of the DADO Tree

relevant to more than one production, portions of WM may in general be replicated in different WM-subtrees.

During the match phase, the WM-subtrees are used as content-addressable memories, allowing parallel matching in time independent of the size of WM. The upper portion of the tree is used to select one of the matching productions to be executed, and to broadcast the action resulting from this execution (both in O(log P) time, where P is the number of productions). Details of these functions follow.

4. The Matching Phase

At the beginning of the matching phase, all PE's at the PM level are instructed to enter MIMD mode, and to simultaneously (and independently) match their LHS against the contents of their respective WM-subtrees. The ability to concurrently match the LHS of all productions accounts for some, but not all, of the parallelism achieved in DADO's matching phase. In addition, the matching of a single LHS is performed in a parallel manner, using the corresponding WM-subtree as an associative processing device. The simplest case involves the matching of a single LHS pattern predicate containing at most one instance of any variable. In order to match the predicate

(Part-category =part electronic-component),

for example, the PM-level PE corresponding to the production in question would first broadcast a sequence of instructions to all PE's in the WM-subtree that would cause each one to simultaneously compare the field beginning in, say, its fifth RAM cell with the string "Part-category". All non-matching PE's would then be disabled, causing all subsequent instructions to be ignored for the duration of the match. Next, the string "electronic-component" would be broadcast, along with the instructions necessary to match this string against, say, the field beginning in the thirty-fifth RAM location of all currently enabled PE's. After again disabling all non-matching PE's, the only PE's still enabled would be those containing a ground literal that matches the predicate in question. If this were the only predicate in the LHS, matching would terminate at this point. It should be noted that the time required for this matching operation depends only on the complexity of the pattern predicate, and not on the number of ground literals stored in the WM-subtree.

The general matching algorithm, which accommodates a LHS consisting of a number of conjoined predicates, possibly including common pattern variables, is considerably more complex. While space does not permit a complete exposition of the general algorithm, readers familiar with the literature of relational database systems, and in particular, database machines, may find the following brief comments illuminating. First, we note that the set of all ground literals in a

single WM-subtree may be regarded as comprising several relations, each the extension of some pattern literal. Viewed in this way, the general production matching problem reduces to a problem for which Shaw [1980] has proposed, and simulated in software, a highly efficient solution involving the use of associative hardware to evaluate relational algebraic primitives in parallel. The result is a new relation embodying the variable bindings corresponding to all possible instantiations of the production in question that are consistent with the contents of WM.

5. The Selection Phase

Since each production is asynchronously matched against the data stored in its WM-subtree, the production matching phase will in general terminate at different times within each PM-level PE. At the end of the matching phase, the PM-level PE's must thus be synchronized before initiation of the selection phase. In support of this synchronization operation, each PM-level PE sets a local flag upon completion of its own matching task. The I/O switch contains combinatorial hardware that permits the DADO tree to compute a logical conjunction of these flags in time equal to O(log n) gate delays. DADO's tree-structured topology, along with the combinatorial, as opposed to sequential, computation of this n-ary "logical AND", lead to a synchronization time which is dominated by that required for matching, and which may, in practice, be ignored in analysis of the time complexity of the production system cycle.

The selection of a single production to "fire" from among the set of all matching productions also requires time proportional to the depth of the tree. Unlike the synchronization operation, however, the primitive operations required for selection are computed using sequential logic. We assume that each PM-level PE performs some local computation prior to the synchronization operation that yields a single, numerical priority rating. PE's containing matching productions are assigned positive values, while other PM-level PE's are assigned a priority of zero. We also assume that each PM-level PE has a distinct PE tag, stored in a fixed location within its local memory, which may be used to uniquely identify that PE.

After synchronization, all PM-level PE's are instructed to enter SIMD mode. Each such PE is then instructed to send its priority rating to its parent. Each parent compares the priority ratings of its two children, retaining the larger of the two, along with the unique tag of the "winner". The process is repeated at successively higher levels within the tree until a single tag arrives at the root. This tag is then broadcast to all PM-level PE's for matching, disabling all except the one having the highest priority rating, which remains enabled for the action phase.

6. The Action Phase

At this point, the "winning" PE is instructed to instantiate its RHS, which is then broadcast to the root. Next, all PM-level PE's are enabled, and the RHS of the winning instance is broadcast to all. The details of the action phase are made more complex by the importance of avoiding unnecessary replication of WM literals within the lower portion of the tree, and of reclaiming local memory space freed by the deletion of such literals. These functions are based on associative operations similar to those employed in the matching operation.

The PE's at the PM level are instructed to enter MIMD mode and to concurrently update their WM-subtrees as specified by the RHS of the winning instance.

First, the PM-level PE's perform an associative probe for each literal to be deleted from WM, enabling only those PE's in the WM-subtrees whose local memories are to be reclaimed. The enabled PE's are then instructed by the PM-level PE to overwrite their stored ground literal with a special free-tag identifying empty PE's. This tag is the target of the subsequent associative probe executed for each of the ground literals to be added to WM.

When processing an asserted literal, the PM-level PE first determines whether or not the literal is relevant to its stored production. Next, the associative operation identifies those relevant literals which are not present in the WM-subtree, and thus are to be stored in some empty PE.

After probing for the free-tag, all PE's are disabled except the empty PE's. To avoid duplication of asserted literals, all but one of these PE's is disabled by a multiple match resolution scheme which uses combinatorial hardware in the I/O switch to very rapidly clear a flag in all but an arbitrary "first" enabled PE. The asserted literal is then broadcast to the one enabled PE.

As in the matching phase, the action phase in general will terminate at different times in each PM-level PE. After synchronization, another cycle of production system execution begins with the production matching phase.

7. Specialized Production Systems

The general scheme for production system execution on DADO can be extended to support commutative production systems, as well as "cooperating expert systems" based on multiple, independently executing production systems.

A commutative production system allows each of the matching rules on every cycle of operation to

be selected for execution. The same combinatorial hardware used in the action phase to select a single arbitrary "free" PE supports this operation by enumerating each of the matching productions in an arbitrary sequential order. Each of the RHS's so reported to the root are then processed by the action phase.

In our exposition of the general production system algorithm, it was assumed that the upper tree was rooted at the (physical) root of DADO (see Figure 3-1). Since each PE in the DADO tree can execute its own independent program, the upper tree can be rooted at an arbitrary internal node of DADO. Thus, multiple, independent production systems are executed on the DADO machine by rooting a forest of upper trees at the same fixed level of the DADO tree. Communication among these independent production systems is implemented in the same fashion as communication among the PM-level PE's during the action phase of the (commutative) production system cycle.

REFERENCES

Davis, Randall and Jonathan King.
An Overview of Production Systems.
Technical Report, Stanford University Computer Science Department, 1975.
AI Lab Memo, AIM-271.

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

Flynn, Michael J.
Some computer organizations and their effectiveness.
IEEE Transactions on Computers, 948-960, September, 1972.

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

Fuhlrott, Oskar.
Bibliography on AI Machines.
SIGART Newsletter (79), January, 1982.

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

Rychener, Michael.
Production Systems as a Programming Language for Artificial Intelligence Research.
PhD thesis, Department of Computer Science, Carnegie-Mellon University, 1976.

Shaw, David Elliot.
A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives.
Technical Report STAN-CS-79-778, Department of Computer Science, Stanford University, October, 1979.

Shaw, David Elliot.
Knowledge-Based Retrieval on a Relational Database Machine.
Ph.D. thesis, Department of Computer Science, Stanford University, August, 1980.

Shaw, David Elliot, Salvatore J. Stolfo, Hussein Ibrahim, Bruce Hillyer, Gio Wiederhold and J. A. Andrews.
The NON-VON Database Machine: A Brief Overview.
Database Engineering 4(2), December, 1981.

Stolfo, Salvatore J. and David Elliot Shaw.
Specialized Hardware for Production Systems.
Technical Report, Department of Computer Science, Columbia University, August, 1981.