DADO: A Parallel Computer for Artificial Intelligence

Salvatore J. Stolfo

Associate Professor

Department of Computer Science

Columbia University

6 June 1985

Index terms: DADO, Fifth Generation Machine, OPS5, Parallel Processing, Production Systems, PROLOG.

A considerable amount of interest has been generated recently in specialized machine architectures designed for the very rapid execution of AI software. The Japanese Fifth Generation Machine Project, for example, promises to deliver a device capable of computing solutions of PROLOG programs at execution rates on the order of many thousands of *logical inferences per second.* Such a device will require high-speed hardware executing a large number of primitive symbol manipulation tasks many times faster than today's fastest computers. This rather ambitious goal has led some researchers to suspect that a fundamentally different computer organization is necessary to achieve this performance. Thus, *parallel processing* has assumed an important position in current AI research. This report outlines the development of a specific parallel machine architecture which has come to be called *DADO* [3, 4]. DADO is a binary tree-structured multiprocessor architecture incorporating thousands of moderately powerful processing elements (PE's). Each PE consists of a fully programmable microcomputer with a modest amount of local memory.

DADO distinguishes itself from other parallel architectures in several ways. First, although DADO is designed as a massively parallel system, the *granularity* (storage capacity and functionality) of each PE remains an open theoretical issue. Studying "real-world" applications executed on a DADO prototype will shed more light on the granularity of a production version of the machine. Second, DADO is designed for a specialized set of applications implemented in Production System (PS) and Logic Programming form. Third, the execution modes of a DADO PE are rather unique. Each PE may operate in *slave* mode whereby instructions are executed as broadcast by some ancestor PE in the tree. Alternatively, a PE may operate in *master* mode by executing instructions from its local RAM. This rather simple architectural principle allows DADO to be fully partitioned into a number of distinct "sub-DADO's", each executing a distinct task.

Finally, DADO has been designed around commercially available, state-of-the-art technology rather than designing everything from scratch. A 15 PE prototype DADO1 machine constructed from Intel 8751 microprocessor chips has been operational at Columbia University since April 1983. A 1023 PE DADO2 prototype is expected to be completed in May 1985.

DADO2 is not viewed as a performance machine, but rather as a laboratory vehicle to investigate fine-grain processors. Although DADO2 is expected to achieve significant performance improvements in AI software (indeed, DADO2 will deliver over 570 million instructions per second), more importantly it will provide a testbed for the next generation machine. The performance of an R1-like rule system running on DADO2 has been studied. Analytical projections indicate that DADO2 can achieve 85 cycles (rule firings) per second using the Intel 8751-based PE design. Present statistics for R1 implemented in a variant of OPS5 executed on a DEC Vax-11/780 indicate from 30-50 cycles per second can already be achieved. Thus, DADO2 performs 50% better than the projected performance of a serial machine much larger and more complex. If a 32-bit PE design were used, DADO2 could be expected to achieve a factor of 16 better performance, or nearly 1360 cycles per second!

Many issues have arisen while studying the granularity question. For example, when the amount of RAM increases, the number of distinct PE's decreases for a fixed size machine, thus reducing the potential parallel execution of code. However, decreasing the RAM affects the size and resultant complexity of code that may operate at an individual PE, thus restricting the scope of applicability of the architecture.

A simple illustration using the R1 expert system may clarify matters. A PS consists of a number of rules which are matched against a database of facts called working memory (WM). As the size of RAM is increased, more rules and WM elements may be stored and processed by an individual PE. However, since fewer PE's are available, less work may be performed in parallel. Conversely, by reducing the size of RAM, fewer rules and WM elements may be located at a PE, but the additional PE's may be able to perform more operations in parallel.

Recent statistics reported for R1 indicate that of a total of 2000 rules and several hundred WM elements, on average 30-50 rules need to be matched on each cycle of operation. Thus, even if 2000 finer-grain PE's were available to process the rules, only 30-50 PE's would perform useful work. Instead, if, say, 30-50 coarser-grain processors were used, each storing many more rules, all of the inherent production matching parallelism would be captured, making more effective use of the hardware.

The advantages of processing WM in parallel have been ignored, however. In a manner analogous to partitioning rules to a set of PE's, WM elements may also be distributed to a set of independent PE's distinct from those storing rules [1, 2]. The grain size of a PE may then directly affect the number of WM elements that may be processed concurrently. Thus, with a larger number of smaller PE's, WM may be operated upon

more efficiently than with a smaller number of larger PE's. It follows that a "tug-of-war" between production-level and WM-level parallelism provides an interesting theoretical arena to study the tradeoffs involved between parallel processors of varying granularity.

However, the reported statistics for R1 are based on a problem-solving formalism that has been fine-tuned for fast execution on serial processors, namely OPS5. *Thus, the inherent parallelism in R1 may bear little resemblance to the inherent parallelism in the problem R1 solves, but rather may be an artifact of current OPS5 production system programming on serial machines.* An alternative approach is to provide other formalisms that allow one to explore and implement much more parallelism than OPS5 encodes or encourages. Towards that end, the development of *HerbAl* (named in honor of *Herb*ert Simon and *Al*len Newell) has been undertaken. HerbAl is a production system language upward compatible with OPS5 but providing constructs to manipulate WM in parallel and execute multiple rules in parallel. HerbAl thus provides additional constructs which make more effective use of the underlying DADO architecture, potentially producing more dramatic speed up of AI computation than may be possible with OPS5, or specialized OPS5 processors. The development of a logic-based programming formalism, called LPS, for a Logic Programming System has also been undertaken. LPS is somewhat similar to HerbAl, but provides a more powerful logical unification pattern matching operation as in PROLOG.

References

[1] Miranker D. P.
Performance Estimates for the DADO Machine: A Comparison of TREAT and RETE.
In *Proceedings of the International Conference on Fifth Generation Computer Systems*. Institute for New Generation Computing, Tokyo, Japan, November, 1984.

[2] Stolfo S. J.
Five Parallel Algorithms for Production System Execution on the DADO Machine.
In *Proceedings of the National Conference on Artificial Intelligence*. AAAI, Austin, Texas, August, 1984.

[3] Stolfo S. J., and D. P. Miranker.
DADO: A Parallel Processor for Expert Systems.
In *Proceedings of the 1984 International Parallel Processing Conference*. IEEE, Michigan, 1984.

[4] Stolfo S. J., and D. E. Shaw.
DADO: A Tree-Structured Machine Architecture for Production Systems.
In *Proceedings of the 1982 National Conference on Artificial Intelligence*. AAAI, Carnegie-Mellon University, August, 1982.