A NOTE ON IMPLEMENTING OPS5 PRODUCTION SYSTEMS ON DADO

SALVATORE J. STOLFO

CUCS-130-84

CUCS-130-84

A Note on Implementing OPS5 Production Systems on DADO[1]

Salvatore J. Stolfo
Department of Computer Science
Columbia University
New York City, NY 10027

2 July 1984

[1]This research has been supported by the Defense Advanced Research Projects Agency through contract N00039-84-C-0165, as well as grants from Intel, Digital Equipment, Hewlett-Packard, Valid Logic Systems, AT&T Bell Laboratories and IBM Corporations and the New York State Science and Technology Foundation. We gratefully acknowledge their support.

1 Introduction

This brief note is written in response to a recent publication, "Implementing OPS5 Production Systems on DADO," published as a Carnegie-Mellon Department of Computer Science Technical Report by Anoop Gupta in March, 1984. Gupta's paper analyzes the performance of OPS5 Production System programs on the DADO parallel computer, a special purpose production system (PS) machine. The analysis leads Gupta to conclude that DADO is not an effective OPS5 PS machine. We have studied Gupta's analysis carefully, and conclude that his conclusions are inaccurate, flawed by, at times, incorrect or outdated information about the DADO2 prototype, and, at other times, inexact reasoning.

We have divided the following into two sections. Section 2 details specific technical errors regarding the DADO2 design cited by Gupta. Although Gupta properly cites statistics we reported in earlier papers proposing DADO, his analysis is based on an earlier design of the second prototype system presently near completion at Columbia University. However, after making the changes appropriate to be consistent with the current technical design of DADO2, Gupta's analysis is also flawed in not adequately understanding the detailed workings of several reported algorithms. Section 3 focuses on philosophical differences. We shall be careful to accurately quote Gupta to strengthen our case that his conclusion is rather weak. We conclude that DADO indeed is an effective OPS5 processor. More importantly, we believe the DADO machine will produce dramatic performance improvements of AI computation when the sequentialities inherent in OPS5 are removed.

The reader should first carefully read Gupta's paper and any one of the most recent reports detailing the DADO system and algorithms, (Stolfo and Miranker, 1984), for example.

2 Details Repaired

One of the stated goals of the DADO project is to understand the nature of the granularity problem in the context of AI production systems. Granularity generally refers to the relative power (function and storage capacity) of a processing element (PE) in a multiprocessor architecture. *Coarse-grain* devices typically refer to multiprocessors consisting of hundreds of large, fast and perhaps specialized PE's each with hundreds of thousands to millions of bytes of memory. *Fine-grain* devices, such as DADO, typically refer to the class of processor consisting of thousands of smaller, less powerful PE's each with a few thousand bytes of memory. *Very-fine grain* is reserved for those processors consisting of hundreds of thousands to perhaps millions of very simple PE's each with very small memories in the range of a few hundred bytes.

Our first published report (Stolfo and Shaw, 1982) proposing the DADO architecture, although uncommitted, referred to the PE design as incorporating a 2K byte RAM. Later (Stolfo, 1983) we revised this figure to 8K as we prototyped hardware and learned more about the nature of programming such a machine. Gupta used the 8K RAM figure appropriately (page 1, last paragraph). However, our current hardware prototyping of DADO2 consists of a 16K byte RAM at each PE and an additional 4K byte ROM. (We are still not convinced we know the proper grain size of the DADO2 PE. During our next year of research experimentation with various application programs running on DADO2 will shed more light on this issue.)

This discrepancy between the 8K and 16K prototype PE affects Gupta's analysis detailing the performance of Algorithm 2, for example. Page 17 details the weak points of Algorithm 2 where he notes the 8K byte RAM is too small. His measurements indicate at times 15K bytes may be required. Thus, the total 20K available bytes are sufficient for Algorithm 2, negating his conclusion. We will return to his analyses shortly.

Another inaccuracy occurs in detailing the DADO2 I/O chip, nearly completed as of this writing. The DADO2 I/O chip is not implemented in custom VLSI (page 3, paragraph 2), but rather in semi-custom gate array employing approximately 1500 gates. Furthermore, we have discovered that the linear ordering of PE's in the tree is irrelevant and hence the planned functioning of the chip has significantly changed. The DADO2 I/O chip provides direct hardware support for Resolve, global Broadcast, Report, memory access with parity, global interrupt with context switch and control of SIMD/MIMD modes of execution of a PE. Tree neighbor communication is supported by direct processor interconnections. DADO2 contains two binary tree interconnections, one interconnecting I/O chips the other interconnecting processors. Hence, if the I/O chip does not function as planned, DADO2 will thus remain operable as evidenced by the functioning DADO1 machine which employs processor interconnections exclusively. Besides providing some measure of fault tolerancy, the two sets of tree connections also admit easy isolation and diagnosing of faults should any arise.

Since we have touched upon the size of RAM of a DADO2 PE and detailed the I/O chip, we should also mention the rationale for using an 8 bit Intel 8751 processor. DADO hardware prototyping began three years ago in 1981. At that time the 8751 was the only commercially available single chip microcomputer in existence that provided 4 parallel 8 bit ports. These ports allowed us to conveniently interconnect a number of PE's directly without additional logic or chips. The 4K onboard EPROM also allowed us to directly program systems software with the opportunity to reburn the ROM when software errors were discovered.

The analyses reported by Gupta should thus be viewed as appropriate only for the prototype, as he correctly observes. Thus, the reported statistics are based on a .5

mip (2 microsecond instruction time), one address, 8 bit PE. Had we initiated prototyping today, we would use a 32 bit two address microcomputer chip with onboard memory, for example the Inmos transputer chip. This design would immediately deliver a 16 fold speed advantage over the 8751, as Gupta notes.

We view DADO2 as a laboratory vehicle to investigate fine-grain processors, rather than a performance machine. Having resisted committing our designs entirely to hardware (the EPROM and the two sets of interconnections) provides us with the means to experiment with a flexible device. We expect DADO2 to produce significant performance improvements demonstrating our architectural principles. More importantly, it will provide a testbed for the next generation machine, which will have the advantage of even better hardware available for its design. Indeed, all of our software is implemented in high level languages to allow easy porting to the newer designs.

We now focus our attention on Gupta's detailed analysis of three algorithms. We shall not pick apart his numbers in terms of the number of effective cycles of execution. The numbers he reports are adequate for our purposes for the time being. Rather, we shall first focus on the number of PE's he calculates as required to implement each algorithm for a variety of OPS5 programs. Here we have found significant errors.

Gupta begins with the original DADO algorithm he calls Algorithm 1, reported in (Stolfo and Shaw, 1982). His version of the algorithm as reported is accurate but flawed by a fundamental misunderstanding of its implementation.

We stated our algorithm in the most simplistic terms for pedagogical reasons bound by publication length restrictions. The stated algorithm is thus a model, and not an implementation. Several crucial changes and enhancements are ignored by Gupta. Thus, he cites, as we did, that the algorithm requires one production per PM-level PE (page 9), and one WM element in each WM-subtree PE (page 10). For large systems such as R1, which incorporates 2000 rules noted by Gupta, he concludes that 2000 PM-level PE's are required, each rooting a WM subtree with 100 PE's (page 13). Hence, R1 requires over 200,000 PE's to execute on a DADO machine using Algorithm 1.

However, the 16K of RAM available in each DADO PE leads one to the conclusion that much more data may be stored in any PE. Thus, many rules may be stored at a PM-level PE as well as many WM elements within each WM-subtree PE. This has been demonstrated in a working program running on DADO1.

This might then lead one to conclude, as Gupta (page 13), that significant performance degradation will result when additional data is packed into a PE. This is not the case if you consider statistics for R1 that Gupta reports.

The last paragraph on page 13 indicates on average about 50 productions are affected by changes to WM (or need to be matched) on each cycle of execution, whereas page 25 notes 35 productions. Other statistics reported for R1 indicate about 30. Thus, with a suitable partitioning of R1's rule base, each of 30-50 partitions (consisting of about 2000/30-2000/50 or 40-66 rules) would match no more than 1 rule against WM on each cycle.

For DADO2, we can thus conveniently store the PM-level at level 6 of the tree with 32 PE's across. Each of these PM-level PE's would store 62 rules, which is sufficient for all of R1's rule base. Thus, the required 2000 PM-level PE's reported by Gupta is not correct.

Furthermore, Gupta's claim that 100 WM-subtree PE's are required per rule is also wrong. Each WM-subtree PE could store a number of WM elements that are *pattern element disjoint*. That is, store only one WM element at a WM-subtree PE that passes some sequence of Rete one input tests. If another WM element passes the same tests, it would be stored in a different WM-subtree PE.

In R1's case, at most 26 WM-subtree PE's would ever be needed to store all of the WM elements potentially relevant to any single pattern in the LHS of a rule (see page 12, footnote 8). Hence, the example DADO2 configuration for R1 has a PM-level with 32 PE's each rooting a WM-subtree of 30 PE's. The upper tree consists of 31 PE's. Then, on average, one rule would be affected by changes to WM on each cycle within a PM-level PE for R1. Similarly, all relevant WM elements would be accessed on one broadcast through the WM-subtrees on each match of a pattern element. (This configuration presupposes that R1's rule base can be appropriately partitioned so that no more than two rules are affected on each cycle within a partition. It is our conjecture that the match paradigm employed in R1 lends itself to a simple and appropriate partitioning scheme based on distributing one rule from each *method* to distinct processors. These technical issues are left for another paper.) Thus, all of the inherent parallelism in R1 is thus appropriately captured by this example configuration using only 1023 PE's, not 200,000!

The algorithm as stated would need to be slightly modified to identify which of the rules in each partition needs to be matched, rather than iterating over all 62 rules in each PM-level PE. This can be done easily in two ways: either by hashing the WM tokens at the PM-level or by associative probing. The former case has been noted and implemented in several early versions of various PS languages. In the latter case, all of the pattern elements of the 62 rules at a PM-level are distributed uniformly throughout the WM-subtree. If there are 5 pattern elements in each rule on average, a total of about 300 such patterns would be distributed in groups of 10 throughout the 30 WM-subtree PE's.

As a WM token is broadcast during the Act phase, at most 10 associative probes would be required to determine the affected rules. Further, thinking can convince one that one broadcast of the new WM token follwed by a high-speed local search in each WM-subtree PE can calculate this result even faster. Daniel Miranker has analyzed a variant of algorithm 1, called TREAT, which uses this approach and has derived execution statistics for R1 on DADO2 running at 85 cycles per second.

In summary, Gupta's analysis of Algorithm 1 is inaccurate on two counts:

- The total number of PE's required to execute R1 is not 200,000. 1023 PE's would suffice. (We note that his hardware utilization is inaccurate for DADO2 using our example configuration.)

- The stated performance of 11 cycles per second (page 13) is roughly a factor of 8 slower than what actually can be achieved using the .5 mip Intel 8751 PE. (See (Miranker, 1984) for a detailed performance analysis which projects 85 cycles per second.)

Next, we consider Gupta's Algorithm 2, mapping a Rete network directly on DADO. We have already noted the discrepency concerning the size of a DADO2 PE memory and the subsequent inaccurate conclusion.

Gupta also notes that a lower subtree need not be used for conflict resolution since on average 2.5 changes to the conflict occur on each cycle. Thus, reporting the changes to a control processor which performs conflict resolution is more efficient in his opinion. DADO2's I/O chip provides direct hardware support for Max-Resolve which, in one instruction cycle, calculates the maximum value of a register stored in all PE's in the tree. Hence, conflict resolution is remarkably fast on DADO2. This adjusts his figures for conflict resolution from 100 machine instructions (step 4, page 16) to 3-4 instructions, thus attaining an overall execution rate of 75 cycles per second, rather than 67.

He also notes that "...the number of processors used is proportional to the number of productions (approximately eight processors per production)...". Using the general scheme outlined above concerning the use of a PM-level PE for multiple rules, and partitioning rules appropriately, a single PE can be used to store many match nodes compiled from many rules. Thus, once again, we need not use a system with over 16,000 PE's as his analysis implies, but rather 1023 would suffice.

We prefer not to discuss Gupta's Algorithm 3, since this requires an architecture much different than DADO. We do note, however, that similar schemes discussed above concerning distributing data structures (pattern elements, for example) as well as partitioning rules and WM elements significantly changes his performance analysis.

Lastly, the reader is encouraged to read (Stolfo, 1984) which identifies five algorithms for the parallel execution of PS programs on DADO. We believe others exist waiting to be discovered. This leads us to philosophical differences of opinion.

3 Philosophical Issues

The analysis reported is quite good and thorough, although the algorithms he uses are flawed. Only three algorithms are explored, however, one of his own invention that is not suitable for DADO. The statistics used in the analysis, calculated from six existing OPS5 programs, also delineates some fundamental differences of opinion.

We recently reported five algorithms that have thus far been invented for the parallel execution of PS's on DADO. Basing his conclusions on only three algorithms is somewhat incomplete. (We are thankful that 35 or 40 years ago Von Neuman machines were not judged solely on the basis of their performance in executing sequential search.)

Furthermore, as Gupta notes (page 24, last paragraph), OPS5 programs do not have a high degree of parallelism. This is a direct result of the *temporal redundancy* associated with OPS programs. That is, the RHS's of rules generally do not have many effects on WM. But my claim is that ***this is an artifact of OPS5 PS programming on serial machines and not characteristic of the problems the PS programs attempt to solve.*** For example, on page 5 Gupta asserts "The [Rete] dataflow graph embodies the parallelism that may be used to perform the match." We note that the ***Rete dataflow graph embodies some of the parallelism which can be expressed within the OPS formalism, and has little to do with the inherent parallelism in the problem being solved.*** Had Gupta studied another formalism which admits the expression of more parallelism (PROLOG, for instance), he might have started with a slightly different mindset, and he might have reached different conclusions.

We prefer to think along the lines of H.T. Kung. Rather than attempt to speed up the inherent parallelism in existing Fortran codes for numerical problems, for example, rethinking the problem specification leads to systolic parallel structures, based on large-scale parallelism, producing dramatic performance advantages over a ring of Cray's. Indeed, a specialized OPS5 processor may not turn out to be a particularly suitable device for dramatically speeding up AI computation.

A simple illustration using R1 might help clarify matters. R1 uses the match paradigm and works hard to sequence through a static collection of subproblems to solve the Vax configuration problem. Suppose the subproblems are:

```
configure processor
configure memory
configure peripherals
configure cables
```

```
layout
final order
```

The OPS5 program forces serialization, and hence "configure memory" does not occur until "configure processor" is complete and all information appropriate for "configure memory" is available. Now, let's think about another approach. Say for example the original customer specification includes information that microprogram store is necessary. Then, when the "configure processor" subproblem is running, it might produce information for the "configure memory" subproblem that microprogram store must be present. If the "configure memory" subproblem can begin executing prior to the completion of "configure processor" a natural parallel pipelining results. The "configure memory" subproblem might produce information useful to the "configure peripherals" subproblem which can begin executing before both of the previous steps have completed. Synchronization might be necessary, but then again there may be considerable "coarser-grain" parallelism that can be implemented in this fashion, with suitable synchronization constraints. The possibility exists.

The OPS5 implementation of R1 provides little information about what subproblems are inherently parallelizable. Indeed all subproblems are carefully handcrafted to be sequentially executed using "control elements". Our thinking is to provide other formalisms that allows one to explore and implement much more parallelism than OPS encodes or encourages.

For example, two simple experiments have been performed on two small OPS5 programs running at Columbia. A few additional parallel constructs were added to OPS5 which simulate the parallel manipulation of WM. The statistics indicate that these slight enhancements to OPS5 produce a *factor of 6-10 fewer* PS cycles of execution to solve the same problem. Similar studies are underway using the ACE expert system (see (Vesonder et al., 1983)). Thus, we can expect that a slightly different formalism will admit much more opportunity for parallelism, and concommitant speed up, than OPS5.

We next turn our attention to Gupta's interpretation of his analysis. Let us only consider Algorithm 2 with a stated performance of 67 cycles per second, although we have noted 75 cycles per second is more accurate. Several remarks made by Gupta seem inconsistent.

First we note, page 8,

```
''We felt that our rough estimates were good enough, that is, the
inaccuracy in our estimates would not cause a qualitative change in
our conclusions.''

''We believe that a factor of two or three difference in such
calculations will not change our final conclusions about the
algorithms or the architecture.''
```

In his conclusions, he cites a *projected* performance of 30-50 cycles per second achievable on a Vax 11/780 for OPS83 (page 13, third paragraph). Other statistics that have been reported for the OPS5 implementation of R1 on a 780 indicate a performance of 2 to 10 cycles per second. Thus, he implies that DADO2 achieves a 50% performance improvement over the *projected* performance (and a factor of six faster than existing implementations) of a serial machine several times larger and more complex, but adheres to his conclusion that this "...is not much higher than what we can already achieve on a current ... uniprocessor." (page 25, second paragraph).

A factor of two or three beyond this increase, however, produces even more significant performance advantages. Rather than achieving only 50% better performance, DADO2 would achieve 200% to 300% improvements over a 780's projected performance! (If we consider actual reported performance of 2-10 cycles per second, DADO2 would outperform a 780 by a factor of 20!) His comments about his method of analysis and interpretation of results seem rather curious[2].

(To maintain our truth in advertising, we of course must note that the reported statistics for DADO2 are projected, since DADO2 as yet does not exist. Indeed, we would be delighted to achieve 10 cycles per second on DADO2 using the current implementation of LISP we have available. The overhead costs of diagnostics, monitoring and LISP related inefficiencies will undoubtedly cause performance degradation by perhaps a factor of 5-10. As we gain experience with the system, we expect to be able to improve the LISP implementation and remove statistics gathering features to achieve our stated performance objectives. This is of course the major reason for building and experimenting with DADO2 in the first place.)

As noted earlier, (page 25) "a uniprocessor with 32 bit datapaths and a .5 us instruction cycle, already has a sixteen fold advantage over a DADO PE..." Thus, had a 32 bit processor been available for DADO prototyping, we can estimate that a DADO2 would execute over 1100 cycles per second. Miranker's statistics of TREAT would then project 85 X 16 or 1360 cycles per second. Thus, a 1023 PE version of DADO using 32 bit processors is 22-26 times faster than the projected performance of a Vax 11/780 and 110-130 times faster than actual reported performance. Might this be enough to produce a suitable performance factor?

Furthermore, the comment on page 25, second paragraph, "Large-scale parallelism

[2]Equally curious is his footnote on page 3. "Note that as the number of PE's increases, the instruction cycle will have to be slowed down to compensate for the extra logic levels...". However, the machine would have to be slowed by *one logic gate delay* when it *doubles* in size. If pipelined communication were employed, *no* delay would result at all as the machine size increases. This is a rather favorable hardware feature, and not a disadvantage by any means.

almost always [?] implies that each processing element is weak..."[3] is negated by the continual advances in VLSI microprocessor technology. Single chip PE's do not incur the significant chip to chip signal delays (due to wire length and capacitive loading of pins) incurred by board-level PE's. As VLSI continues its downward trend in scaling, performance improvements of single chip processors can be expected to outpace the advances made by board-level processors.

Since the tree structure is a remarkably regular and hence inexpensive architecture to realize in hardware, we assert our conclusion:

```
''The DADO design is a very cost effective OPS5 production
  system architecture, designed to produce significant
  performance improvements over conventional machines.''
```

One remaining issue that seems difficult to resolve is how to compare hardware. That is, what metrics should we use when analyzing hardware complexity? Our comparison of DADO2 to a Vax is probably inappropriate.

Should we compare a DADO2 to a Vax 11/750? We do not know the answer to this question. Indeed, Vax's and DADO's are designed for quite different purposes. Thus, Vax's as general purpose devices have much more circuitry than is necessary for DADO.

We have compared DADO2 repeatedly to a Vax 11/750 primarily to give readers a sense of its physical dimensions and complexity. Cabinet dimensions, number of boards and solder joints are roughly the same for a 16 megabyte Vax 11/750 and a DADO2. However, a 16 megabyte Vax 11/780 is roughly six times larger than a DADO2. (We note with interest that if a DADO2 were coupled to a Vax 11/750, we need only produce a performance factor of 2 over the 750 (since hardware is doubled) to remain cost effective. Gupta's statistics show that we are indeed outperforming a 780 and DADO2 is only a prototype device!)

If a coarser grain parallel symbolic processor with 16 megabytes of RAM based on 1980 commercial chip technology existed, we would be delighted to compare DADO2 to it. Such a device, to our knowledge, does not exist, however.

Even so, the fact remains that the DADO2 system is inexpensive to realize in hardware. (Market *retail* costs of all components for a single DADO2 machine is roughly $100,000, approximately the cost of a 16 megabyte Vax 11/750.) Much theory in VLSI design has been developed which supports this assertion for binary tree architectures. Thus, other parallel devices with the same number of bytes of memory, transistors, wires and solder joints as DADO2 should be used for comparison. We believe DADO2 will remain more cost effective to realize in hardware than these other devices.

[3]The question mark added by the author of this paper.

References

Miranker D. P. *Performance Estimates for the DADO Machine: A Comparison of TREAT and RETE.* Technical Report, Department of Computer Science, Columbia University, April 1984.

Stolfo S. J. *The DADO Parallel Computer.* Technical Report, Department of Computer Science, Columbia University, August 1983. (Submitted to AI Journal).

Stolfo S. J. *Five Parallel Algorithms for Production System Execution on the DADO Machine.* AAAI-84, University of Texas, August, 1984.

Stolfo S. J., and D. P. Miranker. *DADO: A Parallel Processor for Expert Systems.* IEEE 1984 International Parallel Processing Conference, Michigan, 1984.

Stolfo S. J., and D. E. Shaw. *DADO: A Tree-structured Machine Architecture for Production Systems.* AAAI-82, Carnegie-Mellon University, August, 1982.

Vesonder, G. T., S. J. Stolfo, J. Zalinski, F. Miller, and D. Copp. *ACE: An Expert System for Telephone Cable Maintenance.* Proceedings of the International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, August, 1983.