CHAPTER 5. EXPERT SYSTEM PERFORMANCE

5.1 The Benchmark Program

Expert system performance is generally measured in number of Logical Inferences per Second (LIPS). Accurate measurement requires a test problem which involves a large number of inferences. One such is the Towers of Hanoi, the classic problem of moving a stack of disks from one tower to another, moving one disk at a time, and never putting a large disk above a small one. The expert system solution involves only a three rules (Matheus 1986):

a) If the current goal is "move a tower of N disks," remove the current goal, and push three new goals: move top disk to temporary location, move tower of N-1 disks, and move top disk from temporary location.

b) If the current goal is "move tower of 1 disk," replace it with the goal "move disk."

c) If the current goal is "move disk," remove the goal and move the disk.

A "goal stack" keeps track of intermediate goals.[1] These three rules[2] will be evaluated repeatedly until the problem is solved (the goal stack is empty). To solve an eight-disk problem, the system must fire 512 rules[3] and make 255 moves.

5.2 Uniprocessor Performance

The expert system invented during this research (and described in Chapter 3) has been christened TEXMEX -- Threaded EXecution Micro EXpert -- to emphasize that it compiles a rule network to executable, threaded[4] code, and that it can run on small ("micro") processors. Several versions of this software were developed as the research progressed.

---

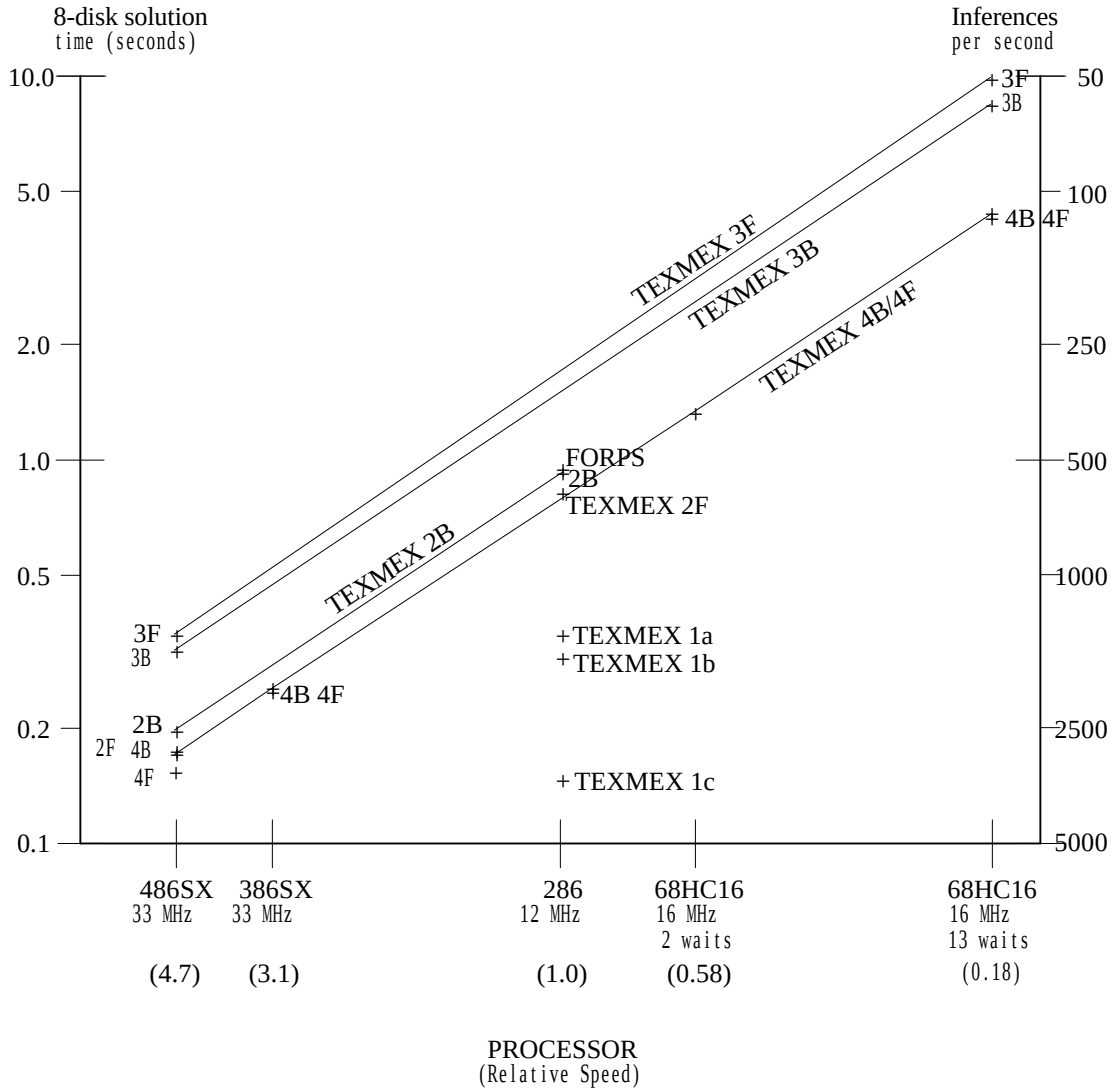
1. In a LISP-based expert system, this stack must usually be simulated with a list. See Appendix F for a CLIPS version of the Towers of Hanoi.

2. Matheus includes a fourth rule which places the original goal on the stack.

3. This number comes from Matheus, who seems to count only firings of rules (b) and (c); it is used here for consistency with Matheus' reported results.

4. See Chapter 3 for an overview of threaded code, or (Kogge 1982) for a detailed treatment.

Figure 5-1 summarizes the performance of this system on a single processor. Data from FORPS, the FORth Production System of (Matheus 1986), is included in this table for comparison.



Notes:

TEXMEX 1a is Direct Threaded Code, with pruning; backward chaining only
TEXMEX 1b is as 1a, with critical sections converted to machine code (hand optimized)
TEXMEX 1c is as 1b, with an optimized goal stack (using Forth's data stack)
TEXMEX 2, 3, and 4 are given for both forward (F) and backward (B) chaining
Processor speed is given relative to a 12 MHz 80286, based on benchmark results

Figure 5-1. Towers of Hanoi Solution Times, One Processor

TEXMEX 1, the first version of the expert system, was written in F83 for the IBM PC. It performed backward chaining only, and compiled the rule network to direct threaded code. Three successive refinements are shown in Figure 5-1. Variant (a) used rule memory and pruning[5] to speed evaluation. In (b), key rule subroutines and logical operators were rewritten in assembly language for improved speed. Finally, (c) dispensed with the auxiliary stack used in the FORPS example to record goals; instead, the Forth data stack was used.

TEXMEX 2, written in Pygmy Forth for the IBM PC, was the first version to allow backward ("/B") or forward ("/F") chaining. It was also the first version to use temporal algebra and expiration times; this prevented "pruning" of the logic tree. This, plus the more complex logic of the inference engine, made TEXMEX 2 about one-third as fast as the comparable TEXMEX 1a. Like TEXMEX 1, rules were compiled to direct threaded code. No speed is lost by forward-chaining; indeed, it is slightly faster than backward chaining.

TEXMEX 3, the first to operate on distributed processors, introduced the Inferencing Token Language "ITL," and compiled rules to a tokenized program rather than direct-threaded code. It was written in F-PC for the IBM PC, and MPE Forth for the 68HC16 microprocessor. The use of tokenized code makes TEXMEX 3 roughly half the speed of TEXMEX 2.

TEXMEX 4 was the object-oriented implementation of TEXMEX 3. The token interpreter and some key methods (e.g. *assert, evaluate*) were rewritten in assembly language for improved performance. This regained the speed lost by TEXMEX 3; even though it still uses a tokenized representation, TEXMEX 4 is slightly faster than TEXMEX 2.

TEXMEX 4 is also twice as fast as TEXMEX 3 on the 68HC16 processor. The speed of this processor was tripled when its internal wait-state generator was reduced from 13 (the reset default) to 2 (the number actually required for the memory in use).

The "inferences per second" statistic was computed from the 8-disk problem. No one system was

---

5. See Chapter 3, section 3.3.

evaluated on all processors; however, enough comparable tests were performed to allow a relative speed metric to be computed for each CPU. These are shown at the bottom of Figure 5-1, with the 12 MHz 80286 arbitrarily[6] used as the "reference" speed.

5.3 Distributed Performance

To test the performance of TEXMEX on a distributed system, a "worst case" test was devised. The Towers of Hanoi problem was divided among three processors as follows:

1) The "tower expert" knows only how to move a tower (rules a and b).

2) The "disk expert" knows only how to move a disk (rule c).

3) The "goalkeeper" keeps track of the current goal (the goal stack).

Thus no processor can perform more than one inference before having to refer to another processor. This is intended to maximize the amount of network traffic, and provide a pessimistic measure of system performance. A "real" problem would be factored so as to *minimize* network traffic, so that each processor could solve, unassisted, the largest possible subproblem.

Figure 5-2 summarizes the results of these tests.

Forward Chaining

At first glance, the reduced performance of the distributed expert system is striking. Using a 486SX/33 MHz, 386DX/16 MHz, and 8088/8 MHz,[7] TEXMEX 3/F solves the 8-disk problem in 44.0 seconds (as compared with 0.353 seconds for the 486 alone). However, instrumenting the network reveals that 2040 packets of 15 bytes each are sent in the solution of this problem. Since the division of tasks ensures that no two sequential messages originate from the same processor, each packet is followed by a two-byte token. In the *best* case (assuming no idle network time) this involves 34,680 bytes, or 36.13 seconds at 9.6 Kbaud. Thus the inference engine is consuming *at most* 12 seconds,[8] as shown in the

---

6. Primarily because all of the early benchmarks used this processor.

7. Three processors of widely varying speeds were chosen to create a more demanding test of the network communications software.

8. More likely, the inference engine is consuming only about two seconds, and network traffic is taking 42 seconds (i.e., a network utilization of 86%).

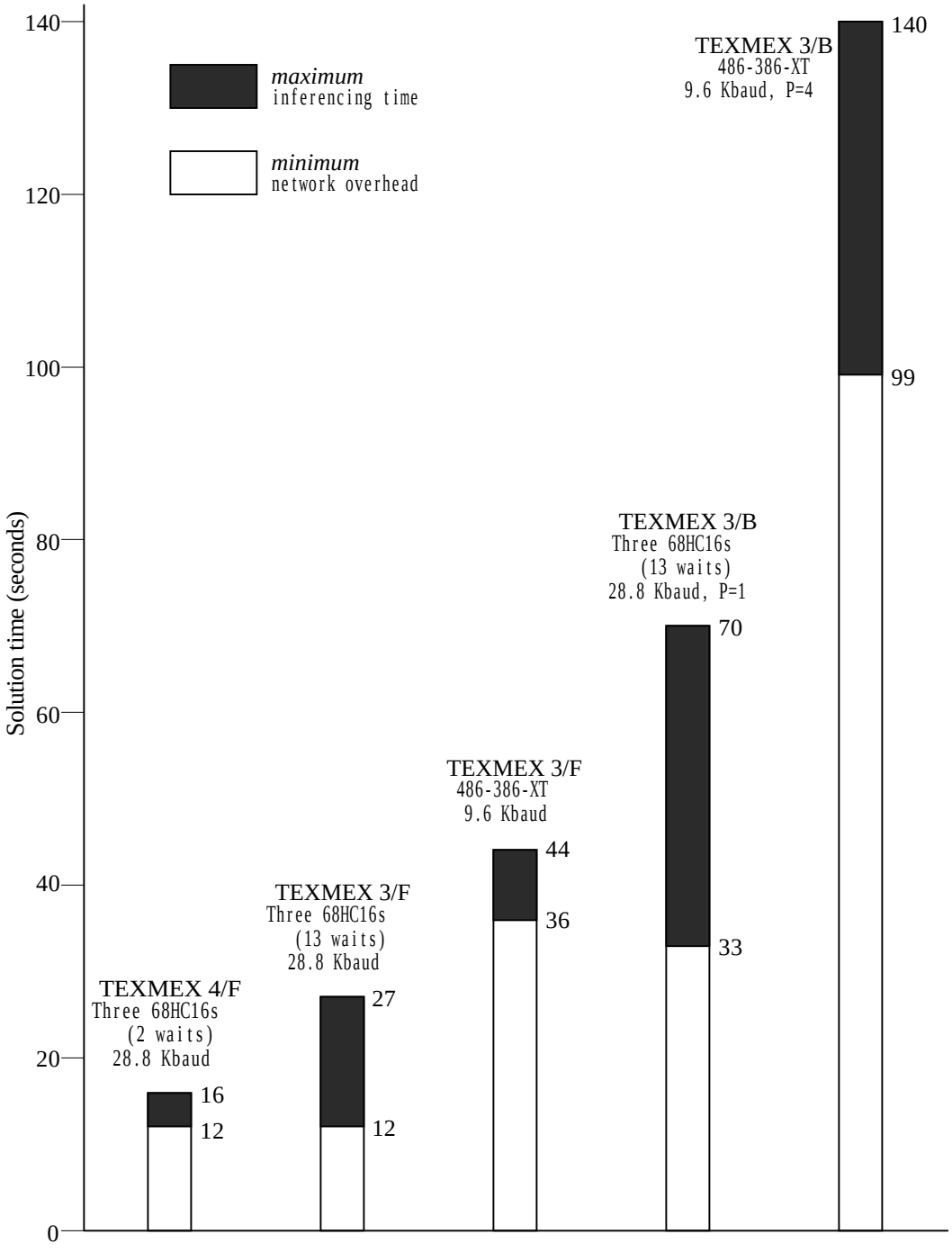


Figure 5-2. Towers of Hanoi Solution Times, Three Processors
(Eight disk problem)

middle column of Figure 5-2. Thus the limiting factor appears to be the network, a conclusion which is confirmed by the tests at 28.8 Kbaud. The first column, for TEXMEX 4/F running on 68HC16s with two wait states, shows a minimum message time of 12 seconds, and a maximum inferencing time of 4 seconds. A single 68HC16 can solve this problem in 1.35 seconds. It is reasonable to assume that the "missing" 3.65 seconds are largely due to less-than-perfect network utilization.[9]

On the 68HC16s, TEXMEX 3/F (13 wait states) solved the 8-disk problem in 26.8 seconds, while TEXMEX 4/F (2 wait states) took 16.6 seconds. This 10.2 second difference is comparable to the 8.6 second difference on a single 68HC16 (10.0 vs. 1.4 seconds). On a 68HC16 with 13 wait states, it appears that about half the total solution time is consumed by inferencing and internal message processing. It appears that the expert system will not be entirely network-limited on embedded processors. Raw CPU performance is still a significant concern; especially if slower (e.g., 8-bit) CPUs are to be used.

Backward Chaining

The backward chaining solution was problematic, since in this case there is no way for one processor to tell the others that a new goal is present. Instead, the tower and disk experts must wait for their current goal to expire, upon which they will query the goalkeeper for a new goal. This expiration time is critical: if too long, the system will spend time waiting idly. If too short, the experts may ask for a new goal before a new goal is available, increasing network traffic with useless messages. Various values of "persistence" time (given in 18.2 Hz clock ticks as "P" in Figure 5-2) were tried. With P=3, repeated 8-disk trials required from 6337 to 7663 packets on the network. The number is more than the expected double of the forward-chaining solution[10], because the backward chaining system cannot use broadcast messages to reduce traffic. The average message length is 13 bytes, plus two bytes for the token; assuming the "best case" of 6337 packets to solve the problem, this represents a message passing overhead of 99 seconds at 9.6 Kbaud, or 33 seconds at 28.8 Kbaud.

---

9. A network utilization of 82% would account for this discrepancy.

10. A doubling of messages is expected because backward chaining requires two messages (ASK and TELL) to transfer a new fact, while forward chaining requires only one (TELL).

Once more, the 68HC16 with 13 wait states spends about half of its time (about 37 seconds) on internal processing. This compares with about 15 seconds for TEXMEX 3/F on the same hardware configuration. Since the single-processor solution times were nearly identical for TEXMEX 3/B and TEXMEX 3/F, it seems likely that the increase in "internal" processing is due to the larger number of messages; that is, message processing (exclusive of inferencing) represents a significant load.

5.4 Comparison to Other Systems

Figure 5-3 compares the performance of the TEXMEX embedded expert system with several other published and commercial packages.

System	Processor	Time (in seconds) to solve 8 disks
TEXMEX 4/F	486SX/33 MHz	0.251
TEXMEX 4/B	486SX/33 MHz	0.258
FORPS	486SX/33 MHz	0.404
FORPS	80286/12 MHz	0.96
TEXMEX 4/F	68HC16/16 MHz	1.36
CLIPS 5.1	486SX/33 MHz	1.42
FORPS	68000/10 MHz	2.31
TEXMEX 4/F	three 68HC16s	16.6
OPS5	TI Explorer	37.8*
REAL-OPS	68000/8 MHz	51.4*
KAPPA PC	486SX/33 MHz	81
OPS5	VAX 11/780	120*
OPS5	MicroVAX II	240*

*this figure extrapolated from data supplied by (Dress, 1986).

Figure 5-3. Comparison of Expert Systems

TEXMEX 4 running on a single processor is two to three orders of magnitude faster than the LISP-based expert system shells, OPS5 and KAPPA, running on comparable hardware. Even the use of a dedicated LISP processor (TI Explorer) does not accelerate OPS5 to one-hundredth of the speed of TEXMEX. Not evident from this table is the fact that the resources required by the commercial expert shells -- megabytes of RAM and a hard disk -- render them useless for embedded control.

The C Language Interactive Production System, CLIPS, is an example of the "state of the art" in real-time expert systems. TEXMEX 4 is almost five times faster than CLIPS. Furthermore, CLIPS requires the resources of a personal-computer: a demonstration CLIPS program compiled to over 155 kilobytes, far in excess of the memory available on an embedded microprocessor.

The only truly comparable system is the FORth language Production System, FORPS. This expert system is well suited to resource-limited processors, and appears only slightly slower than TEXMEX. But since FORPS does not create a network of rules, it must exhaustively search its rule base on every rule firing, with the result that the inferencing speed is inversely proportional to the number of rules defined.[11] The Towers of Hanoi benchmark, having only four rules, is especially favorable to FORPS; a "real" problem would run much more slowly.

5.5 Observations

The statistics quoted for TEXMEX in Figure 5-3 are for a system using a token interpreter. The results obtained for TEXMEX 1 (Figure 5-1) indicate how much more speed is available by compiling to a more efficient code representation (direct threaded code). Compiling directly to machine code would be even faster. "Just in time" compilation gives the benefits of machine code with the portability of tokenized code. The postfix token language ITL is well suited to this, since is can be quickly and easily compiled. This is a promising area for future research.

It has been said that forward chaining is better than backward chaining for process control, since process control is driven by the arrival of new data (e.g. from sensors). From this work, it can now be

11. (Matheus, 1986) estimates 0.7 ms per rule per inferencing cycle, thus a system with 700 rules would perform approximately two inferences per second.

added that forward chaining is also better for distributed inferencing, due to several aspects:

a) A new fact can be transferred with one network message rather than two.

b) A new fact can be broadcast to several CPUs with a single message.

c) Backward chaining requires that queries continually be sent to discover changes in a previously "known" fact. If the query period is too short, the network will be cluttered with useless messages. If the query period is too long, the new fact will not be discovered promptly, and the effective inferencing speed is reduced.

Fortunately, the forward and backward chaining techniques devised in this project achieve equivalent inferencing speeds. There is no incentive to use the "wrong" chaining method for the application when real-time performance is critical.