Section II -- The VAX CPU

The Components of CPU Time

The speed of the VAX CPU, as experienced by applications, is determined by two factors:

- The time it takes to process instructions in the CPU
- The time it takes to access memory to fetch or store the data needed by the instructions.

To some extent there is overlap

- instruction "pre-fetch" (780/785/8200) or "pipelining" (85x0/86x0)
- cache write-through or write-back

Memory accesses may not be required if the desired data is in the memory cache.

When a memory access is required, the CPU waits on the access if it is a fetch. These waits do not show as idle or "null" time; the CPU appears busy. These waits may be longer than the normal memory access times due to other activity in the memory controller or on the CPU bus:

- memory controller completing a previously initiated write due to a cache write-through or write-back.
- instruction pre-fetch initiated read
- I/O, which always has priority.

In most VAX applications, the CPU speed is underutilized -- the speed limitation is the memory controller.

Speed of Memory

The relevant speed is that of the memory controller plus bus arbitration time. Each CPU access causes a fixed amount of data to be delivered: 64 bits on the 780/785/8200, 128 bits on the 8600, 256 on the 8550. The approximate speed ratings are:

8550 = 495 nS
8600 = 560 nS
8650 = 384 nS

MEMORY BYTE WIDTH

CPU	Access Time Microsecs	Max Xfer Rate MM Bytes/Sec
780/785("E" memory)	1.2	6.7
8200	1.2	6.7
8550	.5 to 1.2	26.7 to 48.0
8600	.6	26.7
8650	~.4	40.0

8 32 16 16

26.7 → 64.0

To understand these apparently large numbers, they must be considered in context. A VAX 785 can execute (if not slowed by memory) roughly 1.2 million instructions per second. In typical software, each instruction requires, on average, 11 bytes of data (the instruction itself plus any data referenced by the instruction). If the cache hit rate (defined as memory references met by data previously placed in cache due to direct references) is 50%, the demand on memory is 6.6 million bytes per second. Since there are additional demands, notably from I/O, and there is other activity on the SBI which occasionally makes it unavailable for access, the memory controller is clearly saturated.

$\frac{1.2 \times 11}{2} = 6.6$ MB/

$\frac{6.7M}{11} = .61$ MIPS

In fact, because the demand on memory by the CPU is anything but smooth, the machine will run considerably slower than 1.2 MIPS -- generally about .75 MIPS. (If the software in question takes advantage ot the power of the VAX instruction set, this is roughly equivalent to 1.2 IBM 370 type MIPS.)

The most significant variable is the cache hit rate. The 50% rate assumed above is probably optimistic for large VAX workloads.

Cache hit rates are influenced by:

- The number of active users, and therefore, the context switch rate

- The volume of communications activity (VAX cluster, DECNET or other) which increases the context switch rate.

- The type of application. Highly compute bound scientific computational activity generally has a high rate; data processing, office automation and software development does poorly.

- The program design - its "locality" of data access and code usage.

Implications

Stand alone benchmarks are of little value in evaluating VAX performance.

Traditional standardized CPU speed comparison benchmarks are small programs which will stay totally in cache and do very few memory accesses. The 785 is NOT 1.5 times faster than the 780 in most applications.

Measuring resource utilization of an application by running it standalone fails to account for the effect of context switches on cache hit rates.

I/O transfers slow the CPU due to the additional demand placed on memory. On a 780/785/8200, the running time of software will be lengthened by 4% to 10% during times when a disk transfer is active. Therefore, excess I/O transfers should be avoided.

Good program and data locality is essential in programs which will consume large amounts of CPU time.

Minimize context switching

-Do not use DZ's

-Minimize cluster and DECNET activity. 10^2 to 10^3 Bytes.

-Keep QUANTUM high, if possible

-If primarily a batch environment, keep just enough jobs active to keep the CPU busy.

The dual CPU Vax configurations (782, 8300, 8800) have the same memory controller as their single CPU counterpart (780, 8200, 8550). As that memory controller is the major performance limiting factor in the single CPU configuration, it becomes a crippling performance factor in the dual configuration.

Short instruction formats yield much faster code than longer formats because fewer memory accesses are required and less of the cache is tied up with code.

8600/8650/8550: "Pipelining" architecture doesn't pay off if programs have many branches in code. There is usually little one can do about this at the design level, except to write structured top-down code.

A Major User of Compute Time - Terminal I/O

- no front end on a VAX.

Terminal I/O is a major resource drain on the VAX, not because it overloads any I/O channels, but because it requires large amounts of CPU time. For example: if done naively, reading data from a remote device at 9600 baud can consume 75 to 100% of the CPU time on a VAX 780.

There are two sources of CPU time utilization by terminal I/O:

Interrupt and protocol handling

"Intelligent" features of terminal driver

Character Mode vs DMA vs Ethernet

DZ's operate one character at a time -- each byte of data to be transferred requires a separate I/O func-

tion by the CPU. This is expensive -- a terminal operating at 9600 baud requires 960 separate I/O operations per second and generates 960 interrupts. This will use 5 to 7% of the resources of a 780.

The CPU time spent responding to interrupts and issuing I/O's is "hidden", as it is not charged to the user process. It is responsible for most "overhead" on VAX systems.

DMF's DHU's and DHV's operate in character mode when reading data, and in character mode or "DMA" mode when writing data. While in character mode they have a silo which speeds up output operations somewhat. Generally, however, DMA mode is more efficient except for short transfers, where the cost of setting up the DMA transfer exceeds the cost of doing several, but simpler, single byte transfers.

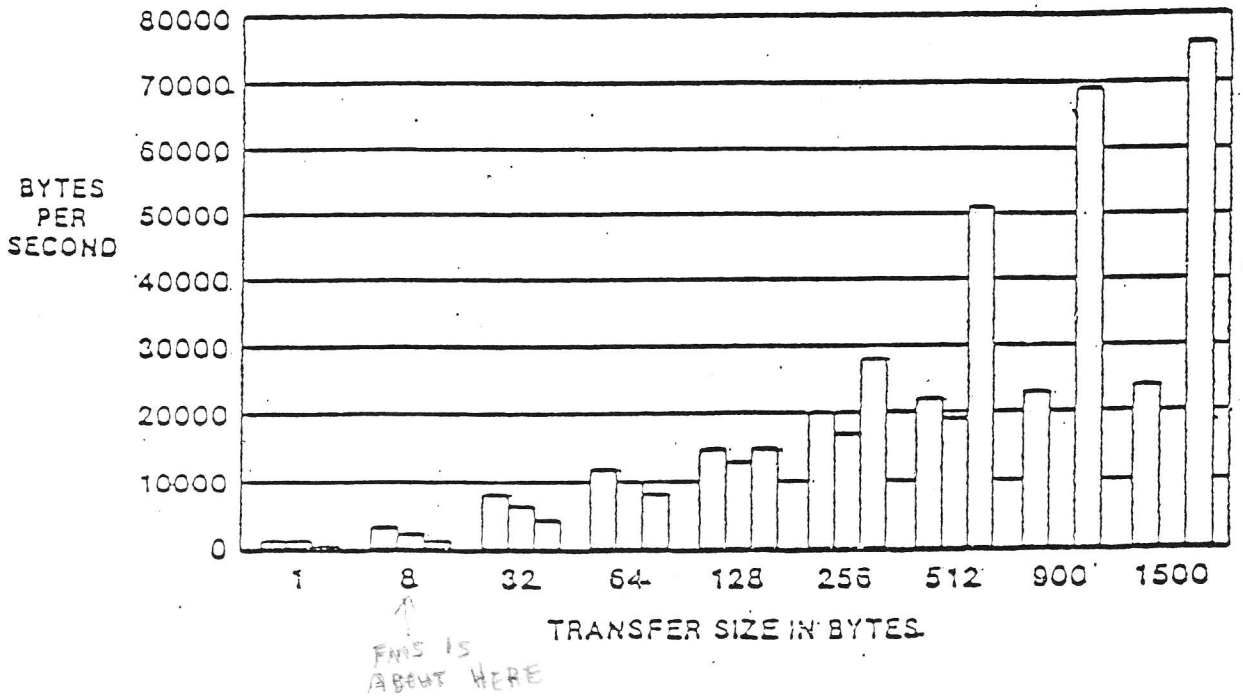
* preferred

The decision about whether to use DMA mode or not on a DMF type device is controlled by the SYSGEN parameter TTY_DMASIZE, which should be set to 30.

See Next Page

For Ethernet devices (the LAT server) all I/O is in DMA mode, and all terminal data is multiplexed (operating on a timeout basis.) The savings over character I/O is largely offset by the cost of protocol handling and multiplexing control.

Figure 2-1
Comparison of Maximum Terminal Output Rates
For Different Transfer Sizes and CPUs

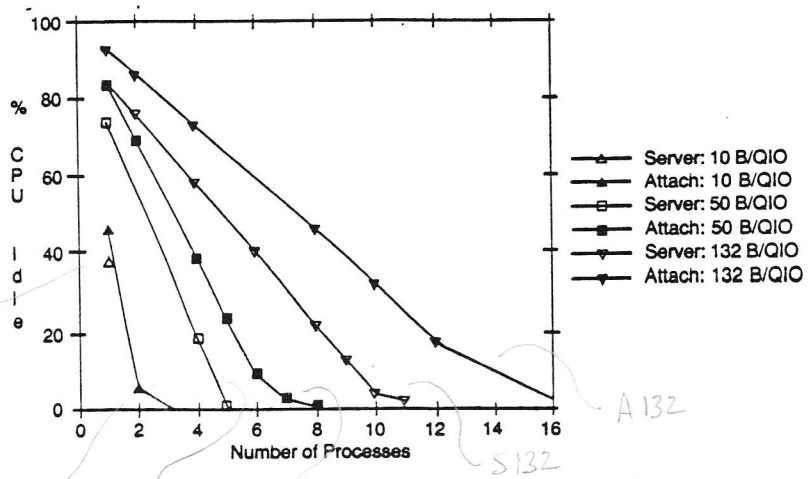


The Y axis represents the data transfer rate achieved at the point of 100% CPU utilization. That is, the CPU processing required just to perform the terminal I/O functions is using all available CPU cycles.

The X axis is the size of the data transfer -- the number of bytes output by each call to QIO. For each transfer size case, there were three tests. The leftmost bar in each case represents a DZ on a 780, the center bar a DZ on a 750, and the rightmost a DMF on a 730. TTY_DMASIZE was set to 64.

TTY_DMASIZE
11/730 - Set to 64 byte
all others - " " 30 "

Figure 2-2
Comparison of Maximum Terminal Output Rates
DMF Type Device vs. LAT Server



August 1985/Hardcopy

The term "Attach" in the legend for the lines refers to the Able Attach terminal interface which is a DMF look-alike, and performs identically to the DMF in all respects.

This plot displays the amount of CPU time unused when doing terminal output at fixed rates with 3 different sizes of QIO. For example, the plot shows that a DMF working at 10,000 bytes per second (10 processes) with 150 character QIO's will use 65% of the system. The LAT, at the same output rate, will use 95% of the CPU.

Terminal Driver "Features" (These are all on by default):

Input - Checks for line editing characters, terminators, etc. Not needed for bulk transmission, block I/O, etc.

Echoes (very expensive - single byte I/O). Needed only for interactive work.

Output- Escape sequence validation, width checking, tab conversions, length checking, etc. These are needed only for debugging purposes. Should be disabled for any high use high terminal output volume production software.

These features are turned off completely by the IO$M_NOFORMAT function modifier when making QIO calls for terminal I/O. High level facilities (languages, RMS) can not turn them off. Efficiency gains of 20% to 80% will follow.

* High level screen management software (TDMS, FMS, the SMG routines) offers possible savings in development and maintenance time and flexibility as to device type. This is obtained at the cost of substantial amounts of CPU time and not necessarily optimal I/O strategies. Users must carefully consider the operational costs versus any actual savings:

- why not hand code most often used 2 or 3 screens and use FMS for the rest.

FMS - 1 I/O per field

Is device flexibility really necessary?

Is the development saving really that large?

Could less broadly functional in-house "library" routines suitable for your flexibility requirements be developed to yield the same application development savings as the DEC routines?

DECNET remote terminal connections (via a SET HOST command) use substantially more resources than any other form of terminal communication. This is due the the high resource demands of DECNET in general, and the fact that DECNET is oriented toward transmission of messages of 100 to 1000 characters as opposed to single character messages.

The PSI X.25 software from DEC has been observed to consume up to 60% of the processing power of the VAX when used for a substantial portion of the machine's terminal connections. Hardware implemented PADs are considerably cheaper and totally offload this activity from the VAX.

PSI software a no-no.
PADS cheaper and better.

Implications

Avoid unecessary terminal I/O:

- Lavish prompts, explanations, graphics, etc. are often unnecessary in software used intensively by users familiar with it. They can, in fact, be an annoyance. (Example of what not to do: compare V4 DCL error messages with V3.)

- High volume printing should be done to devices connected with parallel interfaces.

- Avoid screen repainting if selective updates will do.

- Minimize curser addressing. Use tabs instead of spaces.

- For graphics applications, use intelligent devices and employ their features. (Graphics generation is one of the few applications better performed in local intelligence.)

Keep the number of actual I/O requests to a minimum. Output should be assembled in a buffer, then written 512 bytes at a time, with embedded CR's, LF's and other control sequences. High level I/O routines must be avoided.

Recognize the high cost of screen intensive software. EDT and other screen based editors are costly software to use. Other alternatives may be more productive, both from a machine and user point of view.

Use DMF type devices as opposed to DZ's. Overall, performance gains of 15% are typical. But DMF's won't help very much if I/O lengths are short.

If printers must be driven off terminal lines, DMF's are critical.

The use of the SET HOST facility must be prevented or strongly discouraged.

Avoid data transfers (through DECNET, or with utilities such as KERMIT) over async lines, or confine them to periods of low system activity. If you must do them, use DMF's for output. For input, this application and block mode terminals represent the only areas where the performance of the LAT server will equal or exceed the DMF. Software for async data transfers must be carefully written. Turn off all terminal features, use long fixed length transfers, minimize handshaking and protocol. Use DECNET for convenience, but avoid it for intensive activities.

Ethernet was designed for moving blocks of data (eg., whole documents, facsimiles, files, etc.) among office machines. It was not designed for character by character terminals, even as a multiplexing device. For networks of terminals there is equipment specifically designed for the purpose which is more powerful, reliable, flexible and efficient. If you are using LAT, make the timeouts as long as possible without causing unacceptable response.

Terminal baud rate has no significant effect on the amount of CPU resources consumed to do output -- it simply alters the rate at which those resources are consumed. The only exception is when users of high baud rates are frequently having needed output scroll off the screen before it can be paused resulting in commands being re-executed.

The Xyplex and Telematics terminal interfacing hardware can offload some of the CPU load introduced by terminal I/O. However, much of the same benefits can be achieved through the types of software modifications indicated above. The net gain these devices will produce after such modifications must be carefully in light of their cost. In the case of the Xyplex, if its switching capabilities are to be used, its lack of capability in that regard as compared to the more traditional alternatives should also be considered. (This applies to any local area network based terminal switching. Terminal switches based on wide area networking are generally as inexpensive and effective for local use as those based on a local area network such as Ethernet.)

