UNIVERSITY OF CALIFORNIA

LICK OBSERVATORY TECHNICAL REPORTS

No. 45

THE COMPUTER CHOICE FOR LICK OBSERVATORY

by

RICHARD STOVER

Santa Cruz, California
October 1985

The Computer Choice for Lick Observatory

I. Introduction

Choosing the best computer system for CCD data acquisition, program development and on-line data processing, is not a simple task. We are currently using the Digital Equipment LSI-11/73 CPU. While this is a good CPU, its use has several serious limitations. First, it supports only 16-bit logical addresses, which means that the executable code or data in a single program can total no more than 65-kilobytes. Furthermore it has no additional hardware to support virtual addressing so that various schemes to implement extended addressing tend to be very inefficient. We have expended an enormous programming effort trying to work around this CPU addressing limitation, and the ever increasing CCD dimensions make the task harder and harder to accomplish.

A second serious limitation of the LSI-11/73 is that it uses the Digital Equipment Q-bus backplane. The Q-bus uses only 22-bit physical addresses, so the maximum memory supported is only 4 megabytes. Currently we are using 3 megabytes of memory to store CCD images in floating-point representation so that our SKY array processor can be used with optimum performance. This is enough memory to store only one 800X800 CCD image (2.56 megabytes). With the new 2048X2048 CCD, at least 8 megabytes are needed to store even a single image in 16-bit integer representation, and 16 megabytes are needed to store it in floating-point representation. With today's very low memory prices, 16 megabytes is a very reasonable amount of memory to support.

Another disadvantage of the Q-bus is that it is relatively slow, with the ability to transfer only about 800-kilobytes per second over the bus. This low bus bandwidth is the primary limitation on our ability to rapidly process CCD image data. It places the ultimate limit on the CPU and array processor performance and it limits our Winchester disk throughput, since the disk has the ability to transfer data faster than the Q-bus can handle it.

From our experience with the LSI-11/73 system we have come to realize that a single computer running a single operating system may not be able to fulfill all of the diverse and divergent requirements. We have learned that in CCD data-acquisition and -processing applications we often want dedicated CPU usage to deal with high speed bursts of input from hardware devices concurrently with the multi-user/multiprocess advantages of a sophisticated operating system. The efficient real-time processing of CCD images also requires a large physical memory along with a corresponding large physical address space. In the past, when memory was expensive, virtual addressing schemes were developed which used a large capacity disk storage unit to simulate and substitute for a large physical memory. Today, with low memory prices, it makes much more sense to augment virtual memory addressing with a large physical memory in order to avoid as much as possible the disk input/output bottleneck. With the advent of CCD's which produce 4 million pixels per image, 8 megabytes of memory would appear to be the absolute minimum any new image processing system should support, and two to eight times this amount would be extremely useful.

II. The Choices

There appear to be two classes of machines currently available which may be suited to our needs. The first uses the Motorola 68020 CPU and runs on the VMEbus backplane. We will refer to this combination as the VME/20 computer system. The 68020 is a 32-bit microprocessor which supports a fast floating-point coprocessor, virtual memory management, and 32-bit physical addresses (although most implementations currently use only a subset of this 4-gigabyte address range). The VMEbus is a modern, internationally standardized computer bus designed to support multiple processors, 32-bit data transfers, and a data transfer rate as high as 20 megabytes per second. Furthermore, it is designed for reliable, maintenance-free operation using 96-*pin* DIN connectors instead of the usual failure-prone card edge connectors. The VMEbus, though relatively new in the U. S., is already widely supported by well over 100 firms producing VMEbus-compatible products at competitive prices. We expect the VMEbus to become the dominant bus structure for high-performance super-microcomputers in the near future.

The second candidate for our CCD data-acquisition computer is the Digital Equipment Corporation (DEC) MicroVAX-II. This machine runs the same popular VMS operating system used on Digital's larger VAX machines and, under certain circumstances, is claimed to have a performance comparable to that of a VAX-11/780. Its floating-point arithmetic is particularly fast.

III. The Criteria

There are a number of possible criteria which we could use to choose the best computer system for our needs. As the following comparisons will indicate, the VME/20 and the MicroVAX both have certain strengths and weaknesses.

A. Software Support and Development

Since software development always costs much more than the computer hardware, this is a very important criterion. The MicroVAX will use the VAX/VMS operating system. This operating system is widely used in the astronomical community and we are using it on our own VAX-11/780 departmental computer (which was purchased with NSF funds). Most user programs would be written in VAX Fortran-77. Unfortunately, if any device drivers have to be written for CCD controllers and other special hardware they will have to be written in VAX Macro-11 assembly language or VAX Bliss, a proprietary DEC language. There is a large body of user-level software which we could transport to the MicroVAX with little effort, including our own VISTA image processing system.

The VME/20 will run the UNIX operating system. This is also a popular operating system and is used on many VAX machines. This is also the operating system (UNIX Version 7) we are now using on our LSI-11/73 data-acquisition computers. As part of the normal UNIX software we will have Fortran-77, C, and Pascal programming languages available. For this reason, we could transport essentially all of the data-acquisition and data reduction software (written in C) from the LSI-11/73 to the VME/20. We estimate that about 20% of the code dealing with particular hardware devices will have to be rewritten in moving from our LSI-11/73 to a VME/20 system. The entire data-acquisition system would have to rewritten if we went to a MicroVAX VMS machine. We have extensize reduction software written both for UNIX and for VAX/VMS, so the transfer of reduction software is not a major issue. We will definitely save more time and money, and have an operational computer sooner, by using the VME/20 running UNIX rather than the MicroVAX running VMS.

B. Real-Time Support

UNIX is often said to be a poor 'real-time' operating system, while VAX/VMS is said to be a good 'real-time' operating system, and this is cited as a reason why UNIX should not be used to run a data-acquisition system. If good 'real-time' response is defined to mean that an external interrupt is acted upon within a short period of time, and if that period of time is generously defined to be 20 microseconds, then neither operating system can be said to have good 'real-time' response unless the code handling the interrupt is located within the operating system itself. In other words, user programs can not react quickly to external interrupts. We have seldom, in fact, had the need for this type of fast response since we now use small, dedicated microprocessor-equipped controllers to handle most real-time functions.

VAX/VMS does provide a mechanism whereby a user process can supply an interrupt routine or a device driver for a particular hardware device, but several important restrictions apply to its use. First, a special device driver and interrupt routine must still be used in the operating system. These system routines are used to call the user-supplied routines. Second, once configured with the special driver, the device can not be used with normal system calls. Third, user-supplied routines can not initiate direct-memory-access (DMA) data transfers by the device, as is required by most high-speed data devices. Though not explained in the DEC manuals, this appears to be a hardware restriction imposed by the fact that the memory and peripheral controllers reside on different, incompatible busses which require address translation hardware to get data from one bus to the other. VMS does not permit user programs to manipulate this translation hardware. Therefore, in most cases one will still be forced to write a complete device driver and to install it as part of the operating system. Hopefully, off-the-shelf hardware with vendor-supplied device drivers could be purchased, though this may not always be possible. Given these restraints, the question of which system can provide true, real-time response reduces to deciding for which system it is easier to write a device driver and interrupt handler. Because UNIX is a much simpler system, and because, in UNIX, the device driver and interrupt handler can be written in C, and because we have already written several device drivers for our current LSI-11 UNIX system, we would find device drivers much easier to write for UNIX than for VAX/VMS.

Our experience with both UNIX and VMS shows that the real difference between UNIX and VMS is not one of 'real-time' response, but rather one of exclusive CPU use. Under VAX/VMS the process with the highest priority runs ahead of all others and can therefore acquire the exclusive use of the CPU. The normal UNIX priority scheme is somewhat more egalitarian, in that all processes get to run, regardless of their established priority; high priority processes just get to run more often and for longer periods of time. A process cannot acquire exclusive use of the CPU by simply raising its priority. This has been the only major aspect of UNIX which we have found to be less than optimum for a data-acquisition system. We have overcome this by modifying the UNIX priority scheme slightly, so that the highest possible priority runs exclusively. This modification to UNIX was easy to do because 1) UNIX is a relatively simple operating system and the modifications were minor, and 2) the source code for the system is readily available to universities at very low cost. The same is true for UNIX BSD 4.2 (Berkeley Software Distribution) which we would be running on a VME/20 system.

We have found several other functions useful in a data-acquisition system. Among them is the ability to 'lock' a process in memory, so that the process does not get 'swapped' out to the disk. Both UNIX and VAX/VMS support this 'lock' function. We have also found it convenient to be able to time events or wait for specified periods of time to a resolution finer than one second. This can be done under VAX/VMS but it is not normally possible under UNIX. We have also found a need for modes of inter-process communications in addition to the normal UNIX 'pipes'. We have added both the timer and inter-process communications facilities to our LSI-11/73 UNIX system as pseudo-devices (there's no real hardware associated with them). Under BSD 4.2 UNIX and VAX/VMS extensive inter-process facilities already exist.

C. Large Address Space

To deal efficiently with very large CCD data sets the chosen computer must support both a large virtual address space and a large physical memory. If the virtual address space is large but the physical memory small, a very large fraction of the available CPU time (as well as real, elapsed time) will be wasted swapping data blocks on the disk. The MicroVAX II architecture can support an absolute maximum of 16 megabytes of physical memory and the largest enclosure currently offered by DEC can accomodate only nine megabytes. It is not known whether future MicroVAX machines (MicroVAX III?) will support more physical memory. The VMEbus, on the other hand, is designed to accomodate 32-bit addresses (4 gigabytes). However, most manufacturers of 68020 CPU's are currently offering support ranging from 16 megabytes to 256 megabytes of physical memory. The larger physical address space definitely makes the VMEbus a better choice for our applications.

D. Overall Performance

In benchmark tests developed by John Hennessey at Stanford University and presented by Sun Microsystems Inc., the MicroVAX II floating-point performance was shown to be about a factor of two better than that available from the Motorola 68881 float-point coprocessor often used with the 68020 CPU. On the other hand, the integer arithmetic speed of the 68020 CPU is greater than the MicroVAX's. Since the

VME/20 uses a high speed 32-bit bus instead of the 16-bit Q-bus, I/O operations are always faster on the VME/20. Because of the superior bus the actual performance of the VME/20 machine in most real-world applications was found to be equal to the MicroVAX, even when extensive floating point operations are done. The higher I/O throughput becomes especially important when dealing with multi-megabyte images stored on a disk. The competitive nature of the VME/20 market also means that available floating point performance is likely to increase in the near future. For instance, Sun Microsystems has already announced a floating point accelerator which gives their VME/20 system twice the floating point performance of a DEC VAX-11/780 with floating point accelerator.

Though perhaps of less concern to observers than to theoreticians, it should be noted that the Motorola 68881 floating-point coprocessor is a full implementation of the IEEE Standard for Binary Floating-Point Arithmetic (P754) whereas DEC floating-point arithmetic is not. This lack of standardization has been known to cause problems when transporting code from non-DEC machines.

E. Multi-processor Support

It is very probable that in the future the easiest and least expensive way to gain greater computing power will be to add CPU's (i.e. parallel processing) to a computer instead of starting over with a completely new, faster computer system. To effectively use multiple CPU's to process very large CCD data sets the CPU's will have to be 'tightly coupled.' For the highest performance this means that they will probably have to reside on the *same* backplane, share the *same* mass storage peripherals, and have a large block of common, shared memory. The additional time and processing required to ship very large data sets over any transmission system between completely separate computers could be prohibitive. Separate CPU's coupled only through a multiported disk would suffer from disk contention and limited throughput. Of course one could also envision a multi-CPU computer in which one CPU was assigned the sole task of handling I/O over a very fast optical fiber link. Multiple CPU's on a single backplane also provide a very effective way of separating the small number of true real-time tasks from all non-real-time tasks by assigning the two types of tasks to separate CPU's.

Both the VMEbus and the 68020 are designed to support multiple CPU's on one backplane. This is done with a combination of bus-master and bus-arbitration control lines on the VMEbus and by the implementation of non-interruptable read-modify-write instructions in the 68020 CPU. In fact, many 'smart' disk and tape controllers for the VMEbus already utilize some of these features and have their own 68000 CPU's. Manufacturers are also working on multiple-CPU implementations of UNIX. And a separate processor for real-time functions would make it possible to free UNIX, running on the other processor (or processors), from any real-time responsibilities.

Clearly, these are developments for the future. But the VMEbus is already designed to make them possible and we expect to see them become commonplace in the next five years. The MicroVAX does not support multiple CPU's on one backplane and the MicroVAX II cannot be part of a VAX-cluster, DEC's (extremely expensive) implementation of 'tight' CPU coupling.

F. File Structures

UNIX has only two basic ways of storing data onto a disk, as 'cooked' files or as 'raw' files. Cooked files are the normal UNIX files for which UNIX maintains a file directory. UNIX keeps a list of unused physical blocks on the disk, and it assigns these blocks to a cooked file as the file is written to the disk. Usually, the assigned blocks are not located contiguously on the disk so the contents of the file tend to become scattered over the disk surface. This can lead to an appreciable reduction in throughput when reading or writing very large files (like CCD images), because the disk read/write heads must make a large number of motions over the disk surface in order to access the entire file. To obtain the best possible throughput, our current UNIX data-taking programs store CCD images using the 'raw' mode of UNIX disk file I/O. In 'raw' mode data is written directly from the user's program to the disk device without first being buffered through the operating system. The file is written in amounts which equal an integral multiple of the physical disk block size, and the blocks are physically contiguous on the disk. This provides the highest possible efficiency and throughput when reading from or writing to the disk. In raw mode, UNIX does not maintain a directory structure, so our data-taking program maintains a simple directory of CCD images.

VAX/VMS has many different ways of organizing data in disk files, including the ability to create files whose disk blocks are contiguous. Therefore, in this case there is little distinction between UNIX based systems and VAX/VMS based systems. In fact, the most efficient way of maintaining contiguous files under VMS is probably to emulate the UNIX 'raw' mode. This would be done by creating a single, very large contingguous VMS file into which one could store as many CCD images as would fit into the allocated file.

G. Open and Standardized Architecture

One major difference between the MicroVAX computer and one based on the VMEbus is the basic philosophy of the manufacturers involved. In producing the MicroVAX, DEC apparently prefers to produce a machine which is 'closed', with a peculiar hybrid of modified Q-bus and proprietary non-industry standard memory backplane and bus structure. The same closed, proprietary approach is used by DEC in virtually all of the software products offered for the MicroVAX.

One example of the problems, higher costs, and reduced flexibility which the user encounters because of DEC's proprietary software is worth noting. While looking into the possible use of Ethernet to create a local-area-network, we investigated the availability of 'terminal servers', which allow normal asynchronous, serial terminals to be connected into the Ethernet network. We found that there are no second-source terminal servers manufactured which use the DECNET network protocol because it is a proprietary protocol known only to DEC. There are many terminal servers available which use the well known, public TCP/IP network protocol, which is the protocol used on many UNIX systems. The MicroVAX user is then left with two choices: 1) Buy a DEC terminal server which may cost as much as three times the cost of similar second-source products that use TCP/IP, or 2) Try to run both TCP/IP and DECNET on the same MicroVAX (VMS must have DECNET for inter-processor communications).

The VMEbus, on the other hand, is a world-wide standardized bus with fully published bus specifications. Because of this many peripheral manufacturers are producing a wide variety of high performance devices and interfaces for the VMEbus, and the number of manufacturers is growing rapidly. Some VME/20 manufacturers have implemented special high-speed memory busses for even greater system performance. But, unlike the MicroVAX, the VMEbus architecture is designed to support this type of expansion and the memory cards which have the private memory bus can also be accessed directly from the VMEbus. These dual-ported memory cards make multi-CPU computers much more efficient because bus contention occurs only when the CPU's access shared memory or peripherals on the VMEbus and not when they fetch program instructions over their private memory bus.

Because the VMEbus is a non-proprietary standard, peripheral manufacturers also know they will not be sued for producing new products for the VMEbus. (DEC is suing Emulex for making MicroVAX peripheral controllers).

Some people have argued that, in spite of the 'closed' architecture of the MicroVAX, it would be best to purchase a MicroVAX with all peripherals supplied by DEC. They argue that this type of 'single source' computer would be easier and less costly to keep running since DEC would be responsible for the maintenance of the entire machine (assuming DEC maintenance was purchased). However, we do not believe this is true. Our experience with our own maintenance of our LSI-11/73 systems shows that we are quite capable of isolating failing components to the board level, and that failing boards can be returned to the manufacturer for repair. The cost of long term DEC maintenance on small systems is probably more than the cost of spare boards and repairs. This is especially true if DEC maintenance is required between 5 PM and 8 AM, which are the important hours for astronomy. In reality, it is probably not possible to assemble a suitable 'single-source' machine anyway, since DEC does not manufacture everything we need.

The 'open' architecture of the VMEbus also encourages competition among manufacturers and much lower prices result. Exactly the opposite is true if one wished to buy a 'single-source' MicroVAX. For instance, four megabytes of memory for the VMEbus costs less than $5000, and the cost is rapidly dropping. The same four megabytes of memory for the MicroVAX, purchased from DEC, costs $14,000. Of course it is possible to buy much cheaper memory for the MicroVAX if the mythical 'single-source' machine is abandoned.

IV. Alternates

We are not aware of any other widely available type of computer which would meet our needs. We have considered running UNIX, for software compatibility, on the MicroVAX, but this combination seems to be particularly bad. Although DEC UNIX, known as ULTRIX, is a derivative of BSD 4.2 UNIX, the Sun Microsystems benchmarks showed that programs run under DEC's MicroVAX UNIX execute significantly slower than the same programs run under MicroVAX VMS or under Sun's VME/20 BSD 4.2 UNIX. Apparently ULTRIX does not get the same level of attention that VMS does at DEC. Therefore, running ULTRIX, we would be left with an inefficient system running on an inefficient Q-bus backplane.

V. Conclusion

Both the VME/20 and the MicroVAX II are good, high performance machines. But we are not looking for just a good machine; we need a machine that can grow as our needs grow. For our current needs a VME/20 system is clearly the best choice. Software development time and costs will be less because we will be able to transport much of our current UNIX based data-acquistion and -reduction software to a VME/20. We also believe that hardware costs will be less because of the open architecture of the VMEbus and the competitive nature of the VME marketplace. The VMEbus is a clearly superior I/O bus when compared to the modified Q-bus used on the MicroVAX II. And, unlike the MicroVAX II, the VMEbus provides a reliable long-term path for future performance upgrade and system expansion without being tied to a single vendor whose interests may not coincide with our own. We feel that the multiprocessor capabilities of most VME/20 systems is a very important factor for future enhancements. While the MicroVAX II is likely to be around for many years to come, its use of the Q-bus means that it also represents a dead-end as far as future performance upgrades are concerned. The 16-megabyte physical memory limit on the MicroVAX memory bus also makes the machine less suitable for the efficient processing of very large CCD images.

The biggest market for VMEbus computers to date has been in the field of industrial machine control and factory automation. As a result there is already a wide variety of inexpensive VMEbus based peripheral controllers, single-board computers, ROM cards, backplane assemblies, and other components which astronomical instrument builders would find useful in designing small, dedicated instrument controllers. Many companies make single-board computers which use the Motorola 68000 CPU. If such a CPU were to be used for instrument control the software for the controller could be developed on the large VMEbus data-acquisition computer. Furthermore, the instrument controller code could be written mostly in the high-level language C, since C does not depend on the presence of any operating system and since the UNIX object code linker can produce an executable program suitable for storage in ROM. The use of the VMEbus and the 68000 CPU in both the data-acquisition computer and the instrument controllers will mean that technical personnel responsible for maintaining the system will have to learn only one hardware architecture and one programming environment. This should make both hardware and software maintenance easier, less expensive, and more reliable. Lick Observatory is currently planning to use VMEbus based computers for its future instrument controllers.