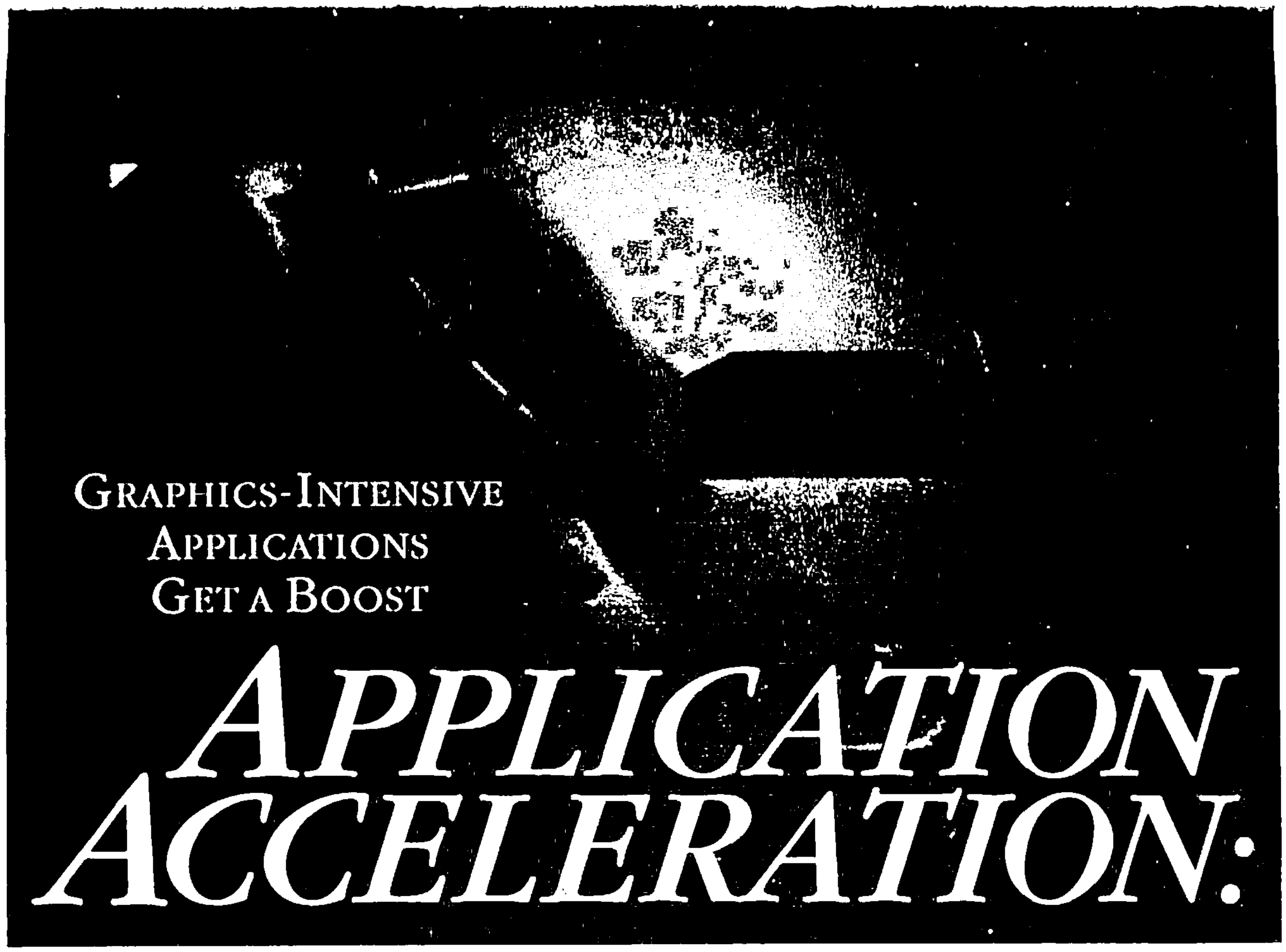


Graphics-Intensive Applications Get a Boost

Application Acceleration: Development of the TAAC-1

By Nick England

The computing engine within a workstation performs numerous tasks—user interface, networking, database management, scheduling, operating-system management—as well as applications. A workstation's CPU (such as the MC680X0 family or Sun's new SPARC chip) is carefully designed to provide rapid context switching and general computing performance. In some cases, however, even higher performance is necessary. To increase overall workstation performance, manufacturers often add specialized processors, such as disk controllers, floating-point processors, and graphics controllers. These specialized devices are each responsible for a portion of the overall workstation task, sacrificing generality for optimized performance.

Issues in Add-on Processor Design

Anyone who designs an add-on processor must face integration issues:

1. How much data passes between the main CPU and the add-on?
2. How much processing per data item will the add-on handle?
3. Is the data flow bidirectional? Or is the add-on primarily a data sink?
4. Will the add-on execute a small set of known functions? Or is it a programmable device?

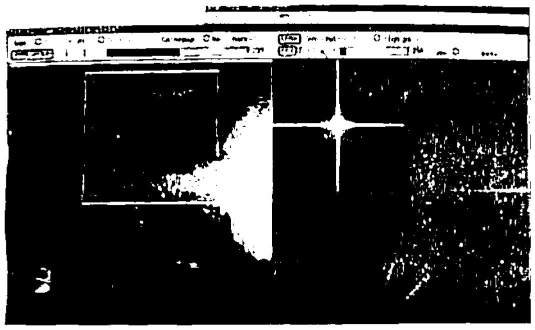
Once these questions have been addressed, the issue of the internal architecture of the add-on processor remains:

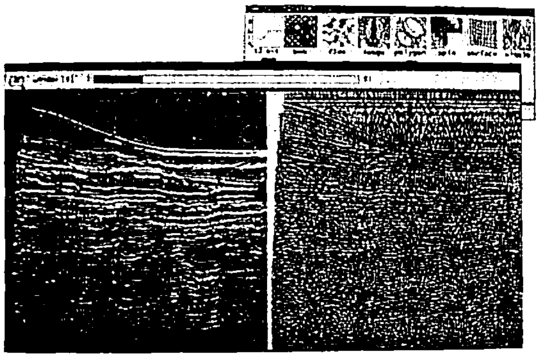
1. What performance level is desired?
2. What flexibility is desired?
3. How narrowly can the types of expected operations be specified?
4. What size task is expected?
5. Are problems typically compute bound or data-flow bound?

Examining the architectures of add-on processors developed for particular applications can lead to the answers to these questions.

A floating-point accelerator (FPA) accelerates individual floating-point instructions and accelerates a limited set of functions or subroutines. In both of these instances, the workstation CPU must load the input-data registers and the appropriate function code and read back the output register.

High-performance graphics are increasingly important for applications such as mechanical design and geophysics.





Architecture

This type of device is characterized by a low data rate (one or two items in, one item out) and a small amount of processing per item. No user programming is provided, and programmed I/O increases efficiency. Typically an FPA add-on is used for tasks such as add, subtract, multiply, divide, sin, cos, and log.

An array processor is more powerful than an FPA in two important ways. First, an array processor can operate with large amounts of data, and it can execute user-developed routines, in addition to standard library functions.

Because an array processor operates on sequences (arrays) of data, a pipelined arithmetic unit is used, which means that start-up involves some overhead, but throughput remains high. The local memory necessary for this kind of operation is typically small (4K-16K words), but it must be fast. This memory is filled or emptied by a direct memory access (DMA) controller, which passes blocks of data to and from the main CPU memory.

This device requires a high data rate (on very structured data) but requires a small amount of processing per data item. Array processors are ideal for use with FFTs, signal processing, and other vector arithmetic.

Image and Graphics Processors
An image processor is similar to an array processor, but it has additional display capability and excludes floating-point data types. Memories are larger (1-12 Mbytes) and organized for fast scan-line access. Random access is available from the workstation bus, but DMA control is usually employed to load/dump images. Like the array processor, this device has a high data rate but low processing complexity for very structured data.

A graphics processor has many elements in common with the other add-on processors discussed above, with an important difference. A graphics processor receives a small amount of data and expands it with a large amount of processing per data item. For example, drawing a full-screen rectangle starts with only a handful of coordinates but results in over a million writes to the video display memory (frame buffer).

A high-performance 3-D graphics processor requires the number-crunching power of an array processor (for transforming input data) and the video data-rate capability of an image processor. Unlike those add-ons, a graphics processor acts as a one-way device, from CPU to screen.

Because of this one-way data flow and the number of functions a graphics processor performs, it usually demands a multiprocessor architecture. One relatively general-purpose processor typically performs command interpretation, a pipelined processor handles transformation computation, and a specialized address generator feeds the frame buffer.

Design Criteria for the TAAC-1 Processor
The TAAC-1 is an application accelerator, a highly programmable device designed to make a class of applications, not just a few selected functions, run faster. It offers display capability, but for tasks that require more processing complexity than an image processor or graphics processor requires. It also has the floating-point performance of an array processor, but with more flexibility. Thus it offers better performance than a general-purpose CPU can provide but does not sacrifice flexibility.

The TAAC-1 was designed for applications involving spatial or geometric data sets. These applications include high-quality graphics, image processing, and analysis. The design of the TAAC-1 architecture took into account the processing requirements of these applications:

1. Large amounts of data are processed.
2. The data are often in 2-D or 3-D arrays.
3. Both integer and floating point are required.
4. Both vector and scalar processing are required.
5. Direct display of results is often required.
6. The processing algorithms are constantly

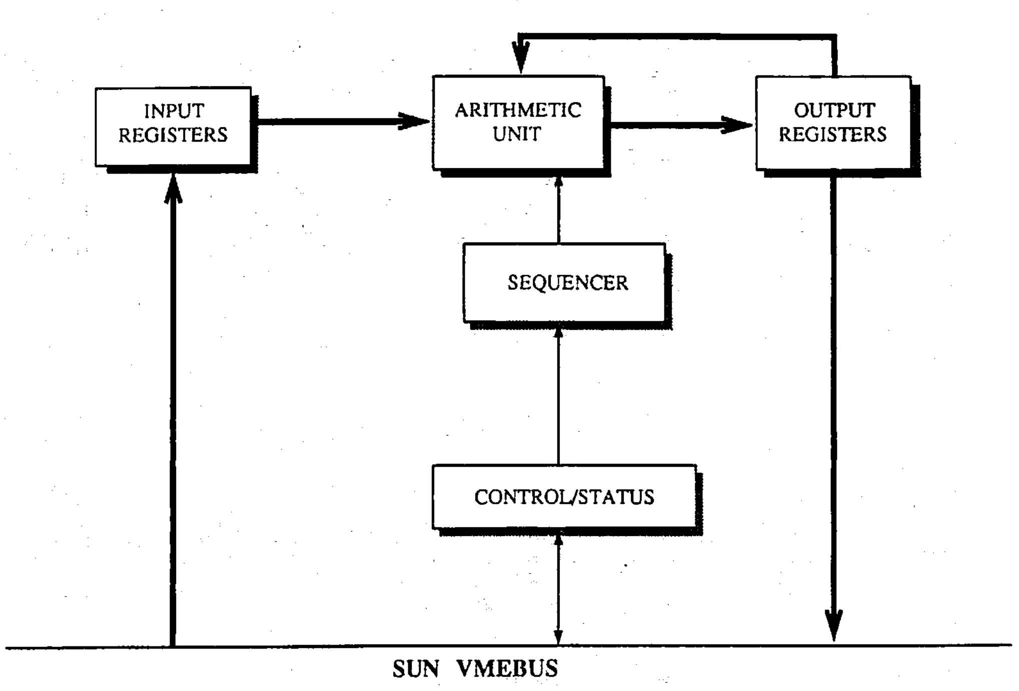


Figure 1. *A floating-point accelerator improves floating-point instructions and a limited number of other functions and subroutines.*

changing, so ease of programming is important.
7. Interactive processing is required.

These considerations led to the following capabilities:
1. A large amount of memory (8 megabytes)
2. Enhanced access to 2-D and 3-D arrays of memory
3. Integer and floating-point processors
4. Vector and random access to memory
5. Display capability from memory
6. Separate instruction memory with C compiler
7. Low-latency, nonpipelined design
8. Memory-mapping into VMEbus

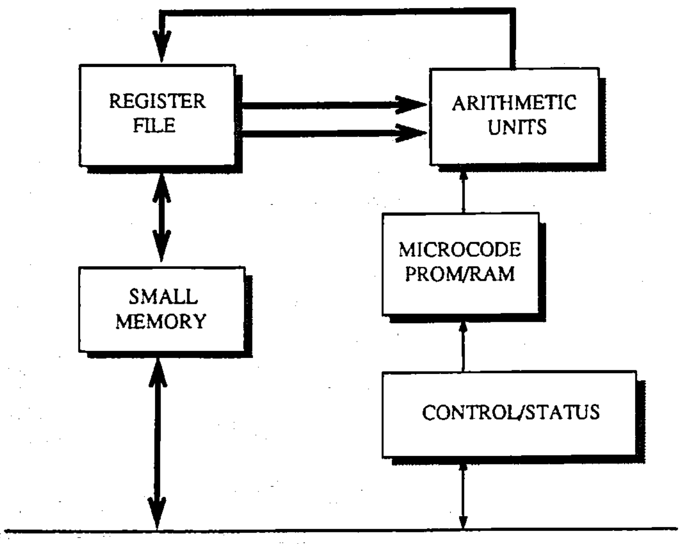
In developing a product with these goals in mind, the TAAC-1 developers were constantly aware of available technology and market requirements. In the case of the TAAC-1, two new technologies were crucial.

Processor technology—Several firms have recently introduced very high-performance 32-bit processors that are extensions of the bit-slice (or horizontal) architecture. These processor families feature high-speed (100 nsec or less cycle time), floating-point as well as integer parts, large register files, and multiple I/O ports. The TAAC-1 uses the Texas Instruments 88XX family of processors, selected after hand-simulation of execution of key algorithms.

The 88XX family has numerous attractive features: It is an extension of a known 8-bit family (74AS888,74AS890). Its power consumption is low. It has several IC technologies available, with some parts already moving from AS to CMOS. Its performance is excellent, and it has a high data throughput. The parts are not internally pipelined and have a horizontal, as opposed to integrated, structure.

Memory technology—The TAAC-1 uses two types of memory, video RAM and static RAM. Video RAM chips are dual-ported dynamic RAMs, which have high serial access rates on the second port. They are typically used to provide video rate data for display refresh. The second

Figure 2. *Array processors are useful for FFTs, signal processing, and other vector arithmetic.*



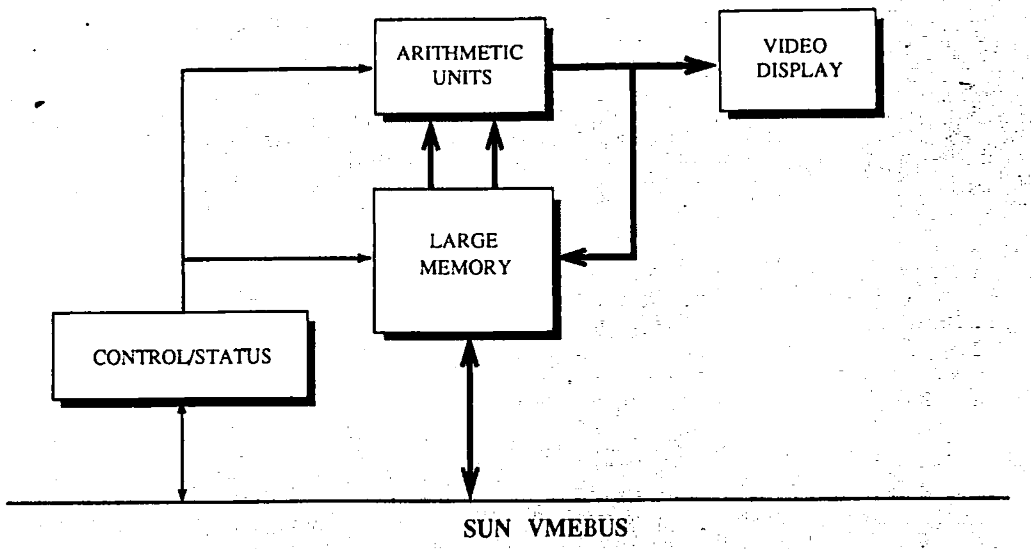


Figure 3. *Image processors are similar to array processors but have more display capability and exclude floating-point data types.*

port can also serve as a high bandwidth port for vector processing. That is, arrays of floating-point numbers can be accessed for number crunching, and pixels can be accessed for display. The memory is divided into two banks, with 400 Mbytes/second vector access to each.

The second element of memory technology involves high-speed static RAMs (SRAM) for instruction storage. The TAAC-1 has a 16K x 200-bit instruction memory using 50 SRAM chips. The SRAMs make possible a very-long-instruction-word (VLIW) architecture, in which each word controls multiple functional units. There are 26 different fields within the 200-bit-wide instruction.

The combination of VLIW architecture and nonpipelined low-latency parts provides very high scalar performance, and the high-memory bandwidth yields high vector performance.

Processor Architecture

The processor section of the TAAC-1 consists of multiple functional units connected by multiple buses. The functions performed by each unit and the sources and destinations of the buses are controlled by fields within the 200-bit-wide instruction word.

The processor architecture was developed with two goals in mind. The first was for application code to run as fast as possible. The second was for compiler development to be reasonably straightforward.

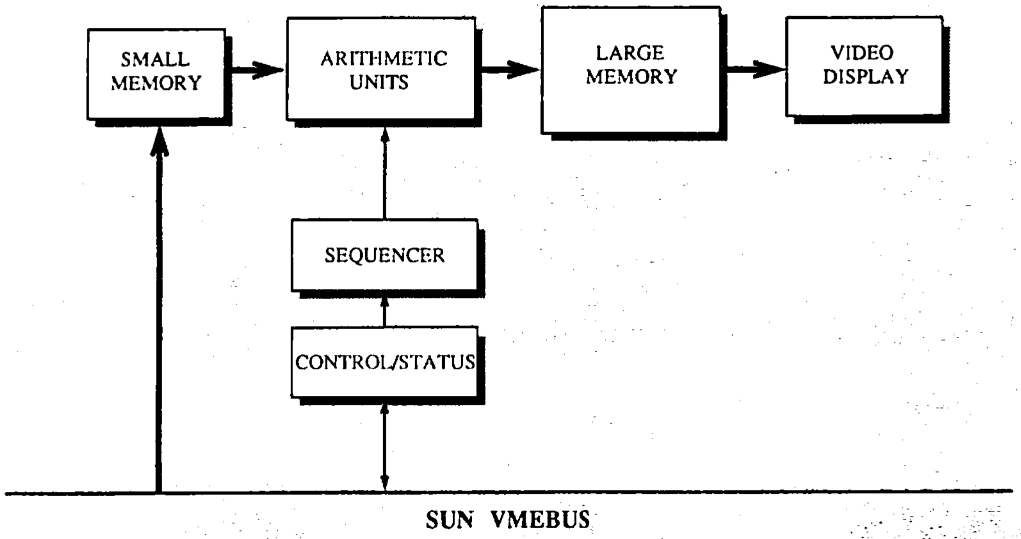
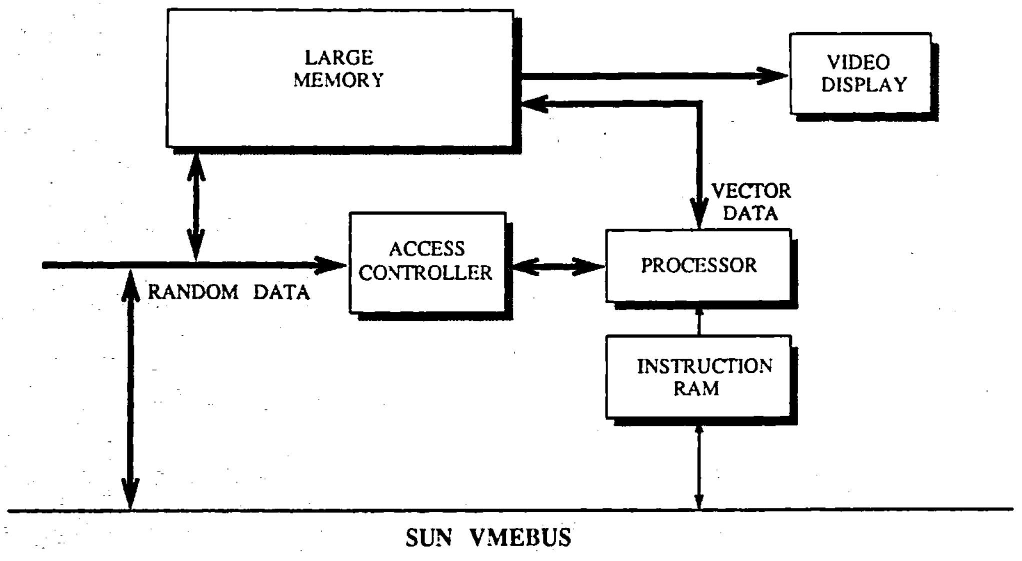


Figure 4. *A graphics processor receives a small amount of data and expands it with a large amount of processing per data item.*

Figure 5. *The TAAC-1 was designed for applications needing spatial or geometric sets.*



The TAAC-1's developers hand-simulated algorithms common in the targeted applications to refine the architecture. The purpose of this exercise was to reduce inner loops to a single instruction, thus providing the same performance that might be expected from dedicated hardware. Two integer ALUs were included, which meant one could be dedicated to address calculations. The integer multiplier/accumulator (MAC) was included for address computation as well as integer data processing. This architecture keeps the free-integer ALU and the floating-point unit supplied with a constant flow of data—an important design goal.

The lookup tables (LUTs) serve two purposes:

1. A preprogrammed LUT provides seed values for Newton-Raphson iteration to compute reciprocals (for division by multiplication) and square roots.
2. A user loadable LUT handles image processing or histogram accumulation.

The access processor helps keep data flowing through the processor. It performs two functions. First, it acts as a smart memory-access controller, ensuring that a programmer is isolated from any concern over memory timing. The controller portion also reduces access time to a minimum in a simple cache-like fashion, so that accesses to nearby regions of memory are faster than arbitrary accesses.

The second function rearranges and controls physical addresses. This function allows programs to access memory in four ways.

1. Linear addressing—2M words (32 bits/word)
2. 2-D addressing—1024 x 2048 pixels (32 bits/pixel)
3. 3-slice addressing—32 256 x 256 slices (32 bits/pixel)
4. 3-D dice addressing—128 x 128 x 128 cube (32 bits/voxel)

The last two methods are ideal for 3-D volume data, such as a series of CT scans or a volume of seismic data. Included within the 200-bit-wide instruction word are fields to control loading, incrementing, and decrementing the X, Y, and Z addresses used in the different modes.

An additional circuit inhibits writes based on a

Table 1. *Summary of various coprocessor attributes and criteria*

DEVICE	ORGANIZATION	DATA RATE	PROCESSING	FLEXIBILITY
FPA	REG ⟶ REG	LOW	SIMPLE	LOW
ARRAY PROCESSOR	MEM ⟶ MEM	HIGH	SIMPLE	HIGH
IMAGE PROCESSOR	MEM ⟶ MEM+DISPLAY	HIGH	SIMPLE	MEDIUM
GRAPHICS PROCESSOR	MEM ⟶ DISPLAY	LOW ⟶ HIGH	COMPLEX	LOW

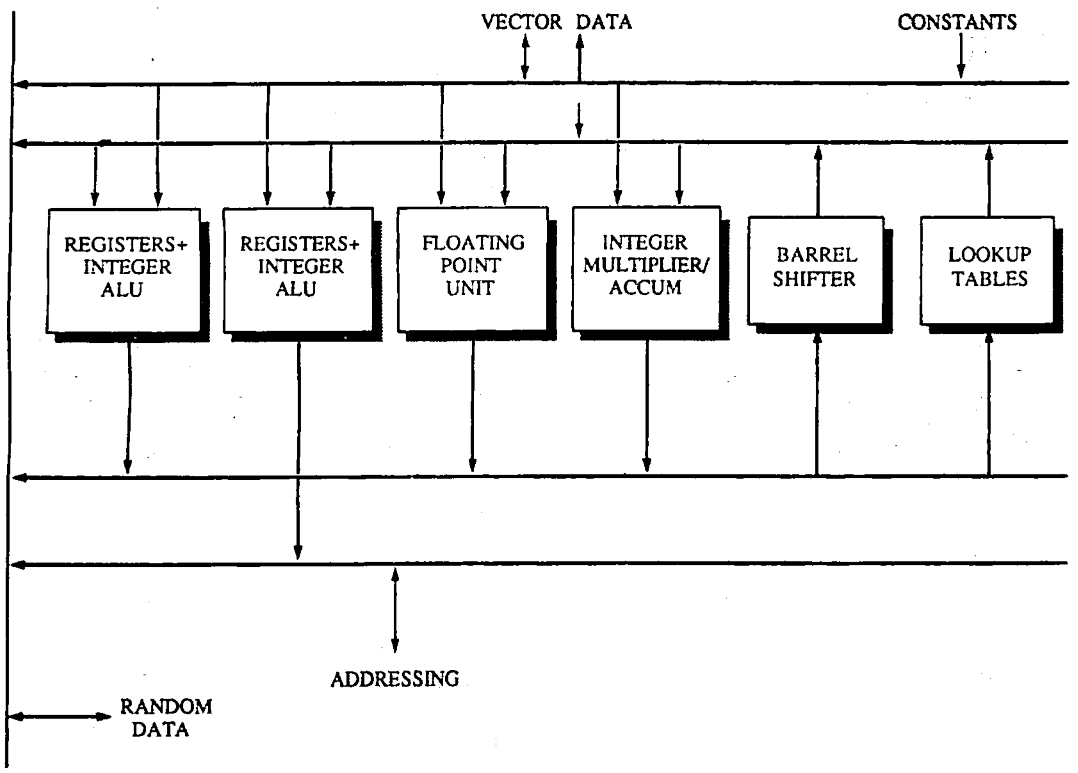


Figure 6. *TAAC-1 processor block diagram*

comparison of the data to be written with the previously read value. This technique is useful for visible-surface-display algorithms.

The TAAC-1's memory uses video RAMs for both display and vector processing. An additional 16K x 32 fast static RAM is also included for stack and global-variable storage.

The data/image memory has a random access port that is typically controlled by the access processor described above. This port is 128 bits wide (multiplexed to 32).

The memory is divided into two banks for vector-port purposes. Each bank has a 128-bit bidirectional bus capable of speeds up to 400 Mbytes/second. The memory array consists of 256 64K x 4 video RAMs (although the architecture is designed to accommodate larger memories in the future). In a typical operation, one bank of memory feeds the display, while the other feeds the processor.

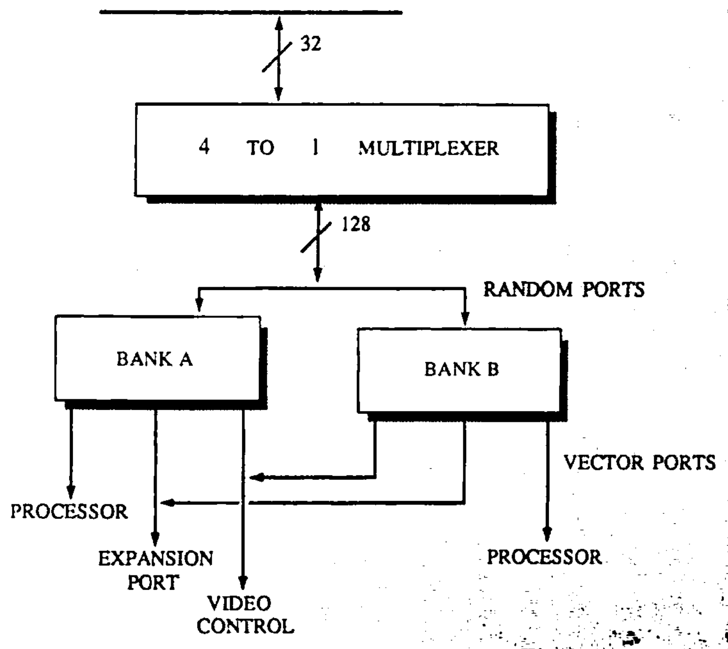
An expansion port reserved for future use is also included in the memory architecture, meaning there are eight 128-bit-wide buses in and out of the memory.

Display-Controller Architecture

The display controller has two separate sections: One controls timing; the other converts digital to analog to drive a color monitor. The timing controller generates a pixel clock and horizontal and vertical signals locked to either an internal oscillator or to an external sync signal. The pixel clock is generated by a phase-locked loop circuit with an extremely wide range (10-100 MHz); the loop is very precise, though (less than 1-ns jitter). The horizontal and vertical signals and auxiliary signals, such as video blanking, are generated from bit maps loaded into RAMs addressed by counters. This arrangement gives maximum flexibility in providing any timing characteristics.

The digital-to-analog section consists of four

Figure 7. *Memory block diagram of the TAAC-1 accelerator*



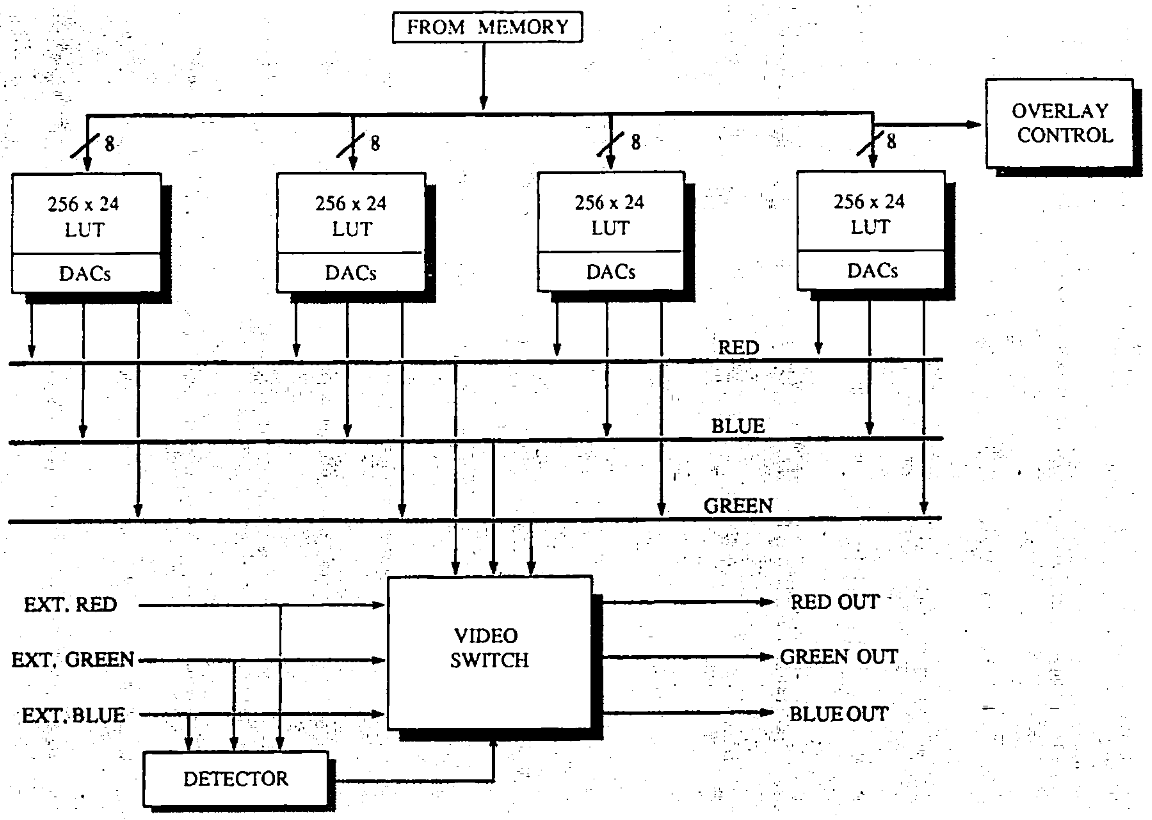


Figure 8.
Video output block diagram

256 x 24 lookup tables driving 12 digital-to-analog converters. Four converters are summed together on each of the red, green, and blue outputs. Again, this design is intended to give as much flexibility as possible in a small amount of circuit-board space. For example, by loading the color LUTs appropriately, a user can switch from a single-channel pseudocolor display (256 colors from a palette of 16.7 million) to a full-color display with 8 bits each of red, green, and blue, plus an 8-bit overlay channel. The overlay circuit functions by turning off the normal display DACs and turning on those belonging to the overlay channel. A bit mask allows user selection of overlay planes. If any enabled bit of a pixel is on, the overlay takes place for that pixel.

Included in the circuitry is a video switch used to insert video generated by the TAAC-1 into an image created elsewhere. For example, it is possible to create a window on the standard Sun color frame buffer and display the TAAC-1-generated image in that window. The video switch circuit responds to a particular color on the input. Whenever that color is detected, the video output is switched so that TAAC-1 video, instead of the externally supplied output, is displayed.

Sun-3 and Sun-4 workstations are ideal environments for the TAAC-1. Like the TAAC-1, their internal VMEbus supports 32-bit data transfers and a 32-bit address. The TAAC-1 is entirely memory-mapped into the Sun address space. Thus, the Sun processor can directly, and with no driver overhead, read and write image/data memory, control registers, and LUTs. This arrangement allows the TAAC-1 to be treated as more memory.

Programming Tools

Because the TAAC-1 is a user-programmable accelerator, it has software as well as hardware, tools. The tools fall into four main areas:

1. assembler/compiler
2. simulator
3. debugger
4. libraries

The assembler, C compiler, and linker/loader were developed by Bit Slice Software of Waterloo, Ontario. The assembler works with separate mnemonic definitions for each of the fields within the 200-bit-wide instruction word. The linker and loader use object modules produced by the assembler. The C compiler is a full implementation of the C language and produces assembly language code as output. Several added functions allow access to hardware features, such as X, Y, Z addressing, that are not readily expressed in C. These special functions set fields within the compiler-generated instructions. Assembly-language code can be placed in-line with C code.

Although a compiler cannot produce code nearly as efficiently as a skilled programmer using assembly language, having a high-level language compiler for the TAAC-1 is an absolute necessity. To begin with, it encourages customers to port code to the accelerator, not rewrite code. Second, software development proceeds much faster with such a tool. Developers almost always develop new library routines, for example, by testing the algorithm in the standard Sun program-development environment, porting that code (through recompilation) to the TAAC-1, and then substituting assembly language code in crucial loops only.

A functional simulator for the hardware tests the programming tools. This simulator allowed developers to have diagnostic routines ready and tested before the hardware was assembled. It also meant demonstration routines were running very quickly. For example, a ray-tracing program (complete with reflection, refraction, and anti-aliasing) was up and running just a few weeks after the TAAC-1 developers received bare prototype circuit boards. Because the TAAC-1 was designed to display images, the simulator also included a bit-map display capability, along with state information on all registers, buses, etc. The simulator performs at 1000 instructions/second (1.00 KIP) on a Sun-3/160, compared to 10,000,000 for the TAAC-1 hardware.

The TAAC-1's debugger allows the display and manipulation of values in all registers and memory locations. This process can be seen in a window on the Sun screen. A programmer can, for example, single-step or set breakpoints because the hardware supports this type of operation. The single-step mode offers single assembly-language steps. The debugger gives symbolic reference to variables and program breakpoints. A complementary profiler graphically displays instruction execution frequency as well as statistical information.

Library routines fall into two classes—those that run on the Sun CPU and those that run on the TAAC-1 processor. The first group comprises useful initialization and control routines and can be linked into user programs running on the Sun. The second group consists of optimized routines for commonly used functions in various [illegible] mon graphics, imaging, and math functions.

Using the TAAC-1

In porting an application or developing a new one with the TAAC-1 accelerator, it is necessary to follow two steps. The first is to identify and break out portions of the application to be moved to the TAAC-1. This step is necessary because the TAAC-1 cannot make any system calls. These separate modules can then be compiled with the TAAC-1 compiler.

An application programmer must then modify the main routines to load data to the TAAC-1 (from memory or disk), start the TAAC portion(s) of the application, and wait for results. These host routines are compiled with the normal Sun compiler. At run time, the two sets of object modules are loaded and executed, one in the TAAC-1 and one in the Sun CPU.

So far, no multitasking control software has been written for the TAAC-1, so dedicated single-task operation is the rule. This method complies with the general modus operandi for users needing an accelerator.

Results

Table 2 shows results from some experimental programs run using the TAAC-1.

This level of performance is generally associated with hardwired, limited-function hardware rather than with a fully programmable computation accelerator such as the TAAC-1.

Acknowledgments

The TAAC-1 architecture and design were developed by Tim Van Hook, who also wrote most of the demonstration and benchmark software. The TAAC-1 prototype was developed at Trancept Systems, Inc., by a hardware and software team including Paul Ramsey, Jamie Blackwell, Mike King, and Marty McLean. Trancept was acquired by Sun Microsystems, Inc., in May 1987 and forms the basis for Sun's North Carolina Application Accelerator Group, part of the Graphics Products Division.

	Sun-3/160	Sun-3/160+ TAAC
Ray Tracing	30 min.	30 sec.
2-D FFT Routines	8 min.	.8 sec.
Adaptive Histogram Equalization	6 min.	4.5 sec.

Library routines for graphics operations give the following results:

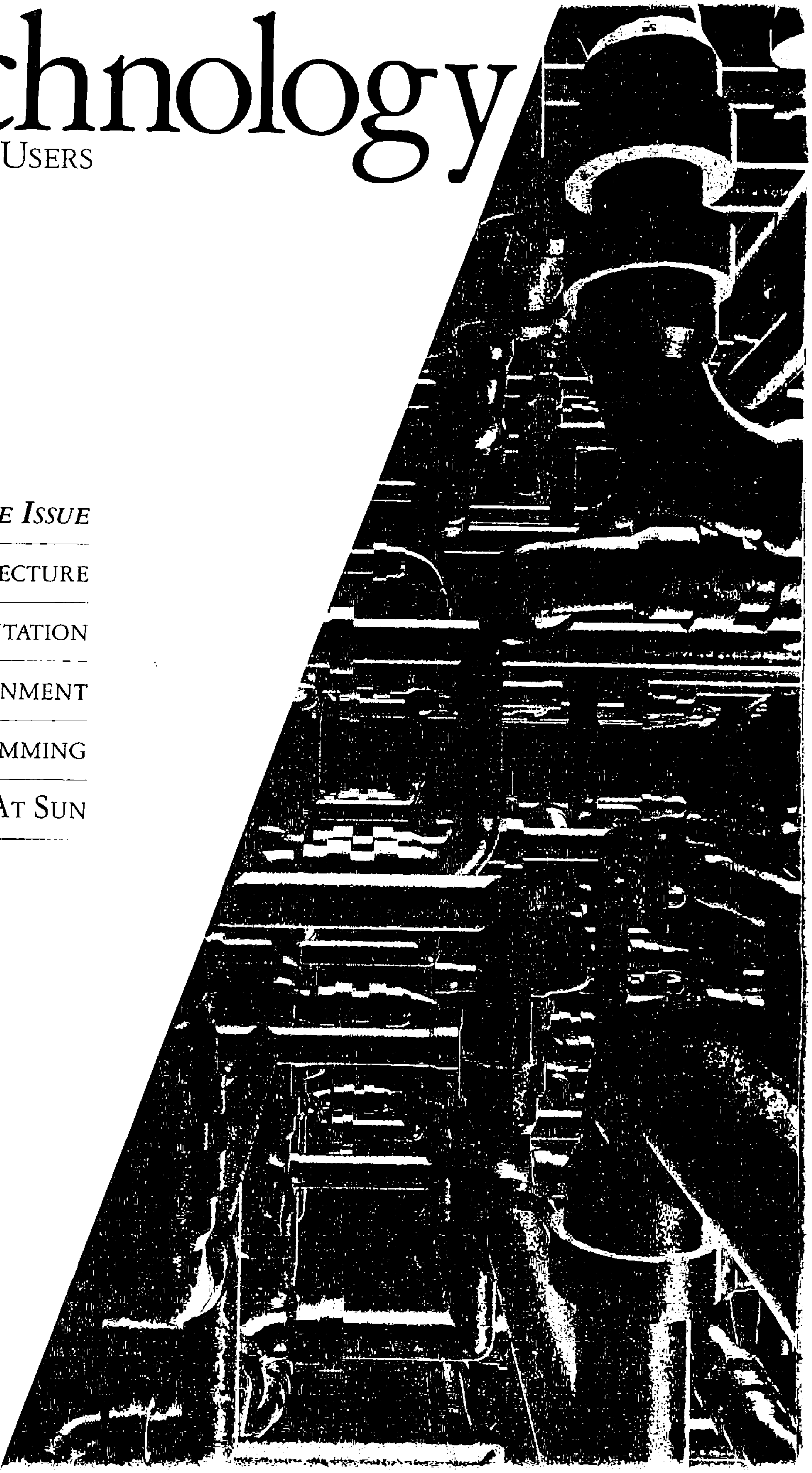
3-D transforms	500,000 per second
3-D vectors	160,000 per second
3-D polygons	30,000 per second

(Gouraud-shaded and Z-buffered)

Table 2. *Results from experimental programs run on a Sun-3/160 with and without the TAAC-1 accelerator*

*Sun*Technology

THE JOURNAL FOR SUN USERS



PREMIERE ISSUE

WINTER
1988