Scaling Solaris for Enterprise Computing

Drew McCrocklin, Cray Business Systems, San Diego, CA

ABSTRACT: *Cray Business Systems chose the Solaris/XDBus platform to develop the Cray Superserver 6400 (CS6400), which is today's largest Symmetric Multi-Processing (SMP) server. Our strategy was to leverage most of the system design and implementation from outside vendors, and to focus our engineering resources on adding value to a base design. We have scaled the hardware interconnect and Solaris operating system to support 64 processors and 16 gigabytes of memory. We added a System Service Processor which supports numerous RAS features, and added a tape subsystem and a math library that are based on widely-used UNICOS features.*

1 Choosing our technology leverage

In November 1991, Cray Research and Sun Microsystems entered into a technology partnership to leverage their complementary strengths across the broader marketplace. As a result of that partnership, Cray formed the Cray Business Systems Division to develop a Solaris/SPARC platform that was scaled larger than Sun's own platforms, and to take the product into markets new to Cray Research. The goal of our division is to develop unique products using primarily leveraged technology. Our plan is to build only the components of the system that provide unique added value.

1.1 Solaris/SPARC strengths

We received four main benefits from the use of Sun's Solaris/SPARC technology base:

1. **The most applications**. Solaris has the largest installed base of any Unix, and has a catalog of over 9,000 native applications.

2. **The best multi-processor Unix**. Sun Microsystems has sold the most multiprocessor systems of any vendor. SunSoft has aggressively threaded Solaris in support of Sun's multiprocessor systems, which scale up to 20 processors.

3. **An openly available Unix**. Sun Microsystems has established a separate company, SunSoft, which has the mission of supplying the Solaris operating environment to the industry.

4. **An interconnect chipset**. Sun Microsystems and Xerox designed the XDBus interconnect chipset for medium and large servers. Sun decided to develop products using only one or two buses, even though the XDBus chipset could handle four buses.

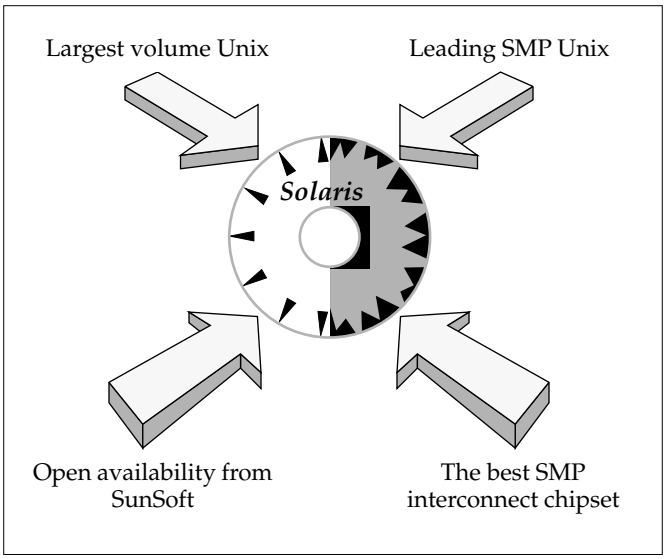

Figure 1: Solaris/SPARC strengths

1.2 Our value-added strategy

We focused our engineering developments in three areas:

1. **Pushing the performance envelope of symmetric multiprocessing**. We implemented the CS6400 using all four of the XDBuses. We designed the centerplane ASICs necessary to drive the four XDBuses across a bigger system, and increased the bus clock from 40 MHz to 55 MHz. With this nearly three-fold increase in memory bandwidth to 1.3 Gbytes per second, we scaled our system size up to 64 processors — the most of any SMP system.

2. **Improving system availability and serviceability**. The CS6400 is built out of a small number of component types which can be configured in parallel to minimize the impact of most system failures. We added the ability to dynamically

reconfigure the CS6400's hardware and software without needing to reboot. We added a service processor to manage the dynamic reconfiguration, and to support diagnostic activity concurrent with system operation.

3. **Adding Cray UNICOS software functionality**. Tape is a traditional Unix weakness, so we chose to add the functionality of the UNICOS tape management system to Solaris. We also optimized the LibSci numerical library for the CS6400 to establish it as a standard for high-performance number-crunching on SPARC systems.

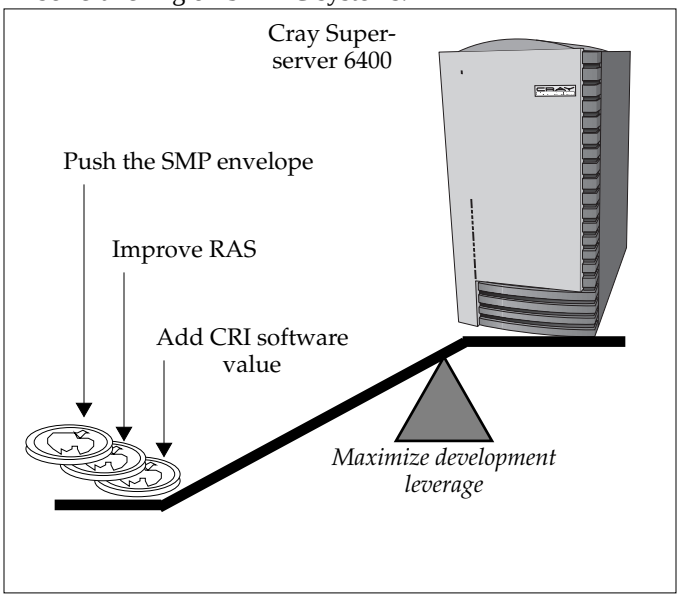

Figure 2: Our value-added.

2 Solaris's kernel modularity

Solaris is a multi-platform Unix operating system that is available on several generations of SPARCs, the Intel X86, and soon on the PowerPC. SunSoft uses one code base for all of these platforms. Their goal is to ensure application portability and compatibility between all Solaris platforms. They must ensure that there is only one set of Solaris application interfaces, and one set of functional interfaces to be used by users and independent software vendors.

To make it possible for independent hardware vendors to add support for new platforms, SunSoft has split the Solaris kernel into two portions. They develop most of the kernel, which is independent of platform. The vendors develop the low-level parts of the kernel that are nearest the hardware, and thus are platform-specific. The division between these parts of the kernel is called the Kernel Binary Interface (KBI). Sunsoft's goal is to maintain this interfaces across multiple platform implementations and releases. Figure 3 shows how the Kernel Binary Interface divides the kernel into the dependent and independent portions.

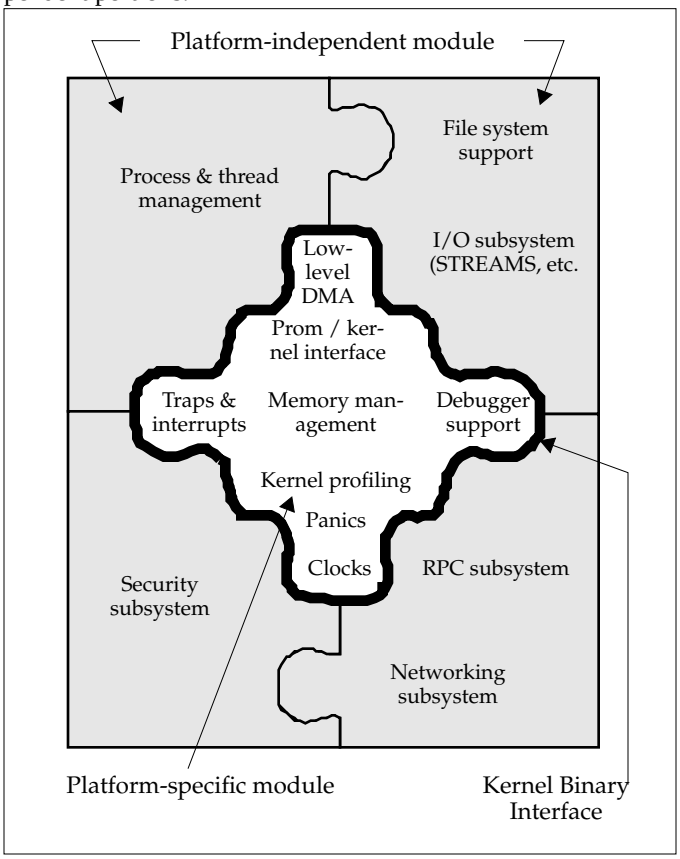

Figure 3: Platform-independent and dependent parts of the kernel.

SunSoft develops and tests the platform independent module, and releases the platform-dependent module for a reference platform. As an independent hardware supplier, we develop and test our platform-specific module for the CS6400, and deliver it to SunSoft. They validate that our module is an accurate Solaris implementation, and ship a CD-ROM containing binary for all the supported Solaris platforms.

The goal is for Solaris on the CS6400 to be the same Solaris as is supplied by Sun Microsystems for its own hardware products. Independent software vendors can then be confident that once they qualify their product on one Solaris platform, that it is qualified on all SunSoft-supported platforms. The CS6400 will be supported by the standard Solaris distribution from SunSoft beginning with Solaris 2.5. Cray's value-added features will then be available for installation as extensions to the standard Solaris base.

3 Enabling large systems

One of our main efforts was modifying Solaris to make it run well on a 64-processor system with very large physical and virtual memory spaces.

3.1 Enabling 64 processors

Scattered through the kernel were constants that had to be increased to reflect a larger maximum system, such as the maximum possible number of processors, SBuses, and boot-buses. We examined all the macros which manipulated bit sets representing individual processors, and expanded the variables from unsigned long to unsigned longlong to contain 64 bits.

3.2 Enabling large memory

The fundamental problem we had in supporting large physical and virtual memory sizes is that stock Solaris maps the kernel address space into the top 512 Mbytes of each 4 Gbyte user context. This mapping speeds system calls, since address spaces do not have to be switched, but it limits the kernel virtual address space to 512 Mbytes.

Our problem was that large CS6400 systems ran out of virtual memory space to hold the tables necessary to manage physical and virtual memory spaces. Solaris was designed to handle conditions where it ran out of physical memory — a condition common on small workstations — but it did not handle well running out of virtual memory, a condition that does not typically occur in systems with less than 4 Gbytes of physical memory.

Room for more physical memory structures. Solaris requires 60 bytes of page structures to represent each 4 Kbyte page of physical memory. As the space needed for these page structures is increased, the *segkp* segment, which is used to map kernel stack space for lightweight processes, is reduced. Using unmodified Solaris, machines that have a large physical memory are limited in the number of processes they can support. At 16 Gbytes of physical memory, 240 Mbytes are required for page structures, so that there is no kernel space left for lightweight-process kernel stacks.

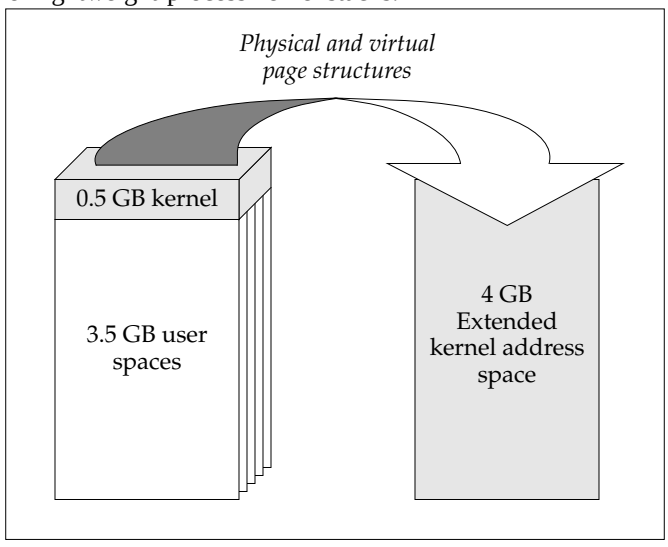

Figure 4: Extended kernel space for large memory.

Room for more virtual memory structures. Solaris requires 26 bytes of anonymous structures for each page of virtual space. For 64 Gbytes of virtual space (a 4:1 ratio over a 16 GB physical memory), 416 Mbytes of kernel space is required, which is far too much for the available 512 Mbytes of kernel address space.

Large memory fix. Our approach to implement large memory is a separate kernel context that leaves the first 512 Mbytes of kernel virtual address space still mapped to all user contexts. During start-up, we relocate non-pageable kernel data structures to an extended kernel address space. The data structures affected include physical memory maps, page structures, page hashtables and memory management unit tables. These structures are accessed by only a handful of routines. Our change frees up substantial virtual address space for the *segkp* segment and more importantly, makes *segkp*'s size independent of memory size. Additionally we have added algorithms that reclaim and manage the use of the kernel heaps in response to running out of virtual memory, in addition to those present to handle physical memory pressure.

3.3 Managing large numbers of processors

We added processor partitions to allow flexibility in workload management. Partitions allow a system administrator to split up the disjoint parts of a workload between groups of processors so that resources are predictably divided among users. Historically this capability was provided in the processor scheduling algorithms with the goal of guaranteeing a certain percentage of each processor to specific workloads. In a cached system with many processors, such as the CS6400, it is much more efficient to dedicate a percentage of the processors to a workload than to dedicate a percentage of *each* processor to a workload. Space sharing is more efficient that timesharing.

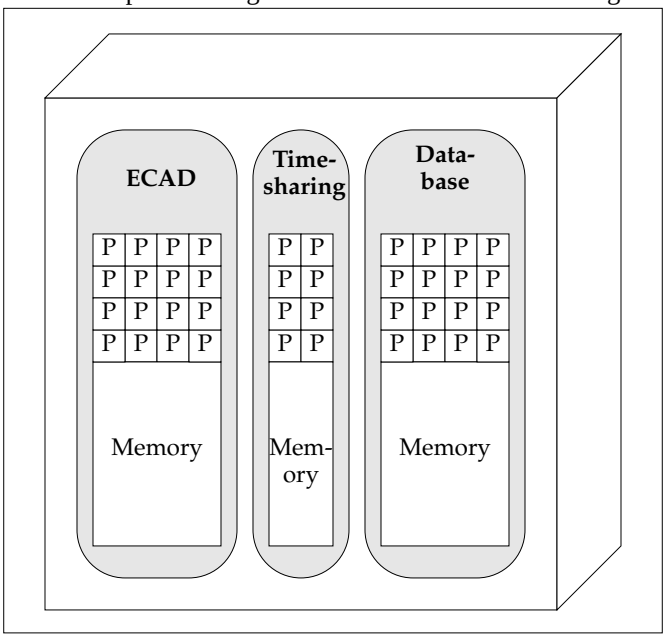

Figure 5: Example of processor partitioning.

The system administrator can set up the partitions, giving them access permissions and symbolic names, and assigning processors and attributes to them. He then assigns particular

process sets to run in particular partitions, or allows users to choose their own partitions from among those for which they have permission.

The processors in a partition give priority to the processes assigned to that partition. Optionally, idle processors can temporarily borrow processes from other partitions, and can loan processes to idle partitions. A partition can be set up to service SBus I/O interrupts, or it can just compute.

The system administrator can execute the partitioning commands dynamically, or can use scripts to partition the CS6400 system during the boot process. She may choose to assign specific processes, such as the NFS server and client daemons, to run in a specific partition. Users may assign themselves to specific partitions as they log in, or may be automatically assigned to partitions based on their user ID or upon partition loading.

We at Cray Business Systems engineering have chosen to divide our CS6400 server into two eight-processor partitions: one for integrated circuit design and simulation, and the other for general timesharing. This split keeps either set of users from hogging the machine's resources. Later this year, the ability to control the amount of memory used by each partition will be added.

4 Tuning large systems

Solaris as we receive it from SunSoft is tuned to the level of 20 processors. This section gives some example cases where we had to make modifications to scale Solaris for larger systems. Some of these problems have been fixed in later releases by SunSoft. In general, Solaris has scaled better with each release as bottlenecks are removed from the parallel kernel code.

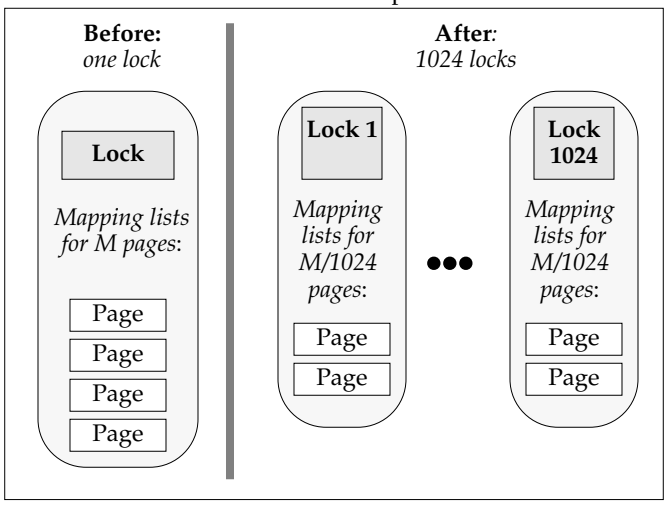

Figure 6: Splitting an address mapping lock.

4.1 Lock splitting to speed-up page creation

We made a significant performance improvement by splitting one of the locks that was used in the Hardware Address Translation (HAT) layer. For each physical page there is a linked list of structures that represent the virtual addresses mapped to that physical page. Each time the Hardware Address Translation layer needed to manipulate that linked list, it had to acquire a global lock used for all physical pages.

We split that global lock into an array of 1,024 locks that we hashed into based on the physical page structure. This made a significant performance improvement during the start-up, page-faulting and exit flows for a process — which was especially important in workloads where processes were rapidly created and destroyed.

4.2 Cache collisions in the idle loop

Idle loop processing. As the Solaris kernel boots, it dynamically allocates a structure for each configured processor. A field in each structure denotes how many runable threads are on that processor's run queue. During the idle loop each processor first examines its own count field, and if zero, it then examines all of the other processor's counts looking for work to do. The dispatcher increments a processor's count when it places a thread on a run queue.

Cache collisions in the idle loop. The SuperSPARC has a direct-mapped cache which is divided into 8192 cache blocks that are 256 bytes long. Memory locations map into the cache using the upper 24 bits of their virtual addresses, modulo the 2-MByte cache size. When the counters from several processors randomly mapped into the same cache block, they displaced each other in the cache, causing several cache misses per idle loop. We noticed heavy bus utilization on partially-loaded systems, where the idling processors consumed enough bandwidth to reduce performance of busy processors by 1/3.

Eliminating the cache collisions. Our solution was to insure that idle processors spin on a single data structure that is known to fit in the cache without mapping conflict. We created an array with a counter for each processor, and spaced the counters out into separate cache blocks.

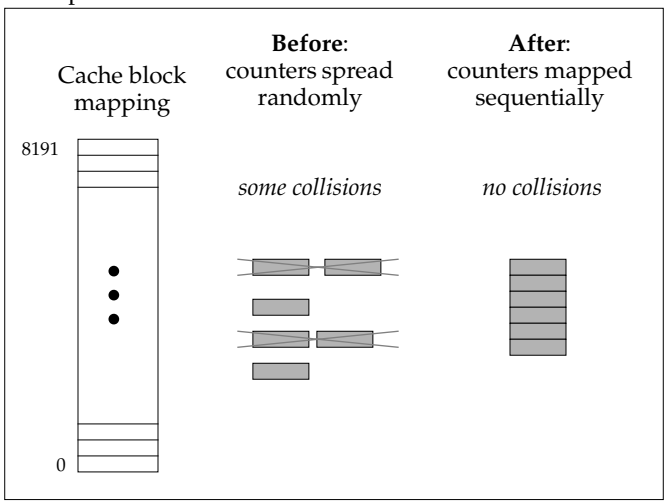

Figure 7: Eliminating cache collisions.

4.3 Cache thrashing during buffer copying

We observed a speed-up of only 2.4x when using four Asynchronous Transfer Mode (ATM) fiber-optic links to transfer

data between two CS6400s. Our investigation showed little evidence of flow-control problems, or kernel lock contention.

We found the most processor intensive portion to be copying data from the user buffer to a system-level streams buffer. When the I/O from a given stream buffer was completed, the stream buffer was placed back into a global pool of buffers for further use by the system. The next stream initiating a transfer would allocate another buffer from this same global pool, and begin the copy from the user buffer to the stream buffer.

We did bus traffic measurements and discovered that the store-miss rate doubled, and the store-invalidation rate quadrupled when going from one to two ATM streams. With four streams running in parallel, there was a 75% chance that a new buffer would still be resident in another processor's cache. Figure 8: shows how a given cache block of data would swap back and forth between the caches of two processors as they recycled buffers from the same global pool.

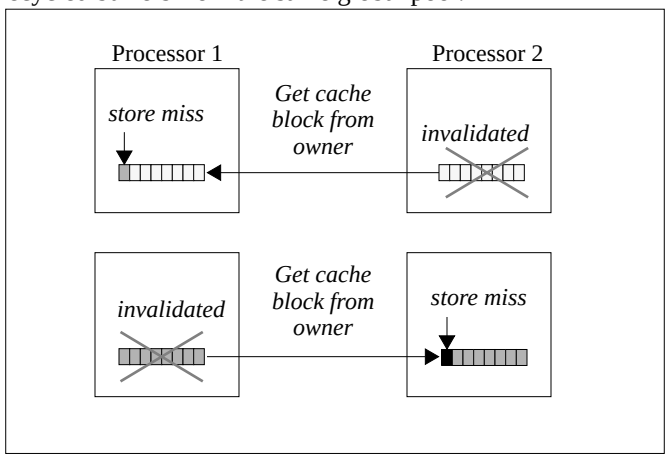

Figure 8: Cache ownership thrashing caused by sharing a buffer pool.

To avoid this needless swapping of cache-block ownership, we used the processor's ID to give each stream its own buffer pool. Solaris 2.4 supports a buffer allocation scheme that allows buffer pools to grow and shrink as demand for system resources changes — which made the creation and management of multiple pools quite reasonable.

Our tuning improved the four-stream bandwidth from 32 to 47 MBytes per second, an increase of 46%, resulting in a parallel speed-up of 3.4x. The curve is shown in Figure 9:.

4.4 TLB missing due to small pages

The Memory Management Unit (MMU) of the SuperSPARC supports three page sizes: 4 Kbytes, 256 Kbytes, and 16 Mbytes. Stock Solaris supports only the small 4-Kbyte size. The SuperSPARC has a 64-entry Translation Lookaside Buffer (TLB) which caches recent address translations. With 4 Kbyte pages, only 256 Kbytes of virtual space can be accessed without a TLB miss, which is only 1/8th the size of the 2-Mbyte cache of the SuperSPARC.

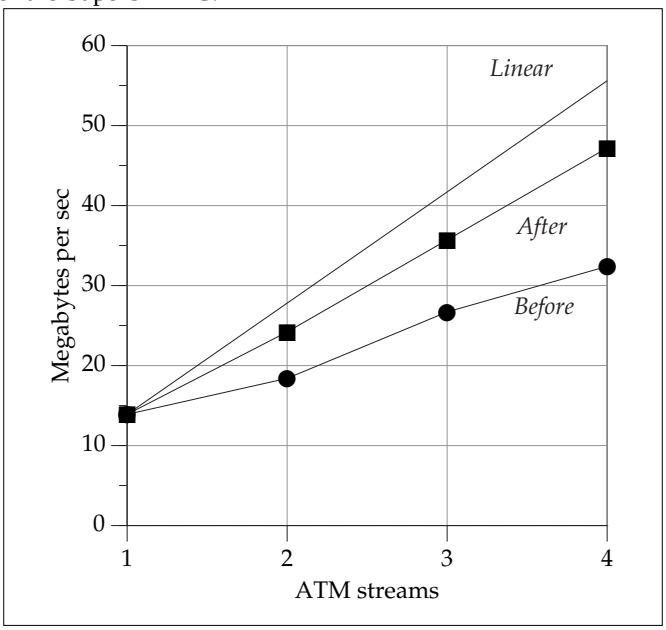

Figure 9: TCP bandwidth using multiple ATM links.

Programs with a working-set size between 256 Kbytes and 2 Mbyte suffer TLB misses even though their all their data has been cached. We chose to support the 256 Kbyte page size, which allows accessing a 16 Mbyte space without a TLB miss. The large page size benefits programs that intensively reference large cache items. For example, we measured a 30% improvement in the performance of calculating 16K FFTs when using large pages. The data all fits in cache, but suffered TLB misses with small pages.

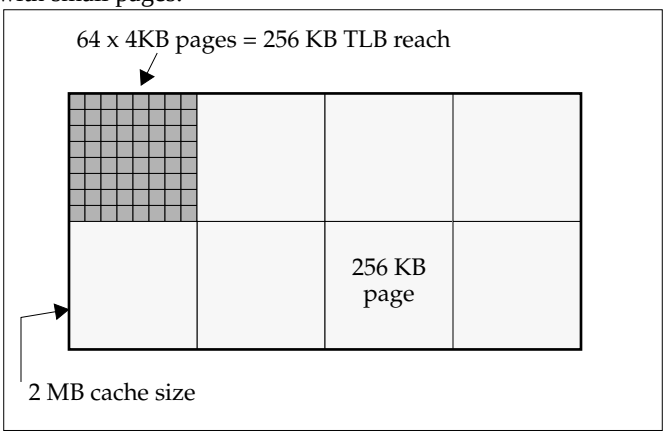

Figure 10: Larger page size for greater TLB reach.

5 Improved availability and serviceability

Reliability, availability, and serviceability are dominant concerns of all large-system customers. As systems become bigger, the impact of downtime expands exponentially — whether it is a commercial customer running an on-line database, or an engineering organization running circuit simula-

tions. The RAS philosophy of the CS6400 is to provide a system that is kept available a very high percent of the time through the configuration and use of functionally redundant components. The availability of the system is enhanced by the ability to dynamically reconfigure the software and hardware and to service the hardware while the system is still operating.

The CS6400 is implemented from many parallel copies of a relatively small number of different component types. There can be up to: 16 power supplies, 64 processors, 64 memory modules, 8192 SIMMs, 16 SBuses, and 64 SBus interfaces. When one of these components fail, the system will automatically not use the bad component, and rapidly reboot to resume system operation. The failed component can be replaced later while the system is operational. Optionally redundant hardware (a form of spares) can be configured to be used as extra capacity until needed to replace failed units.

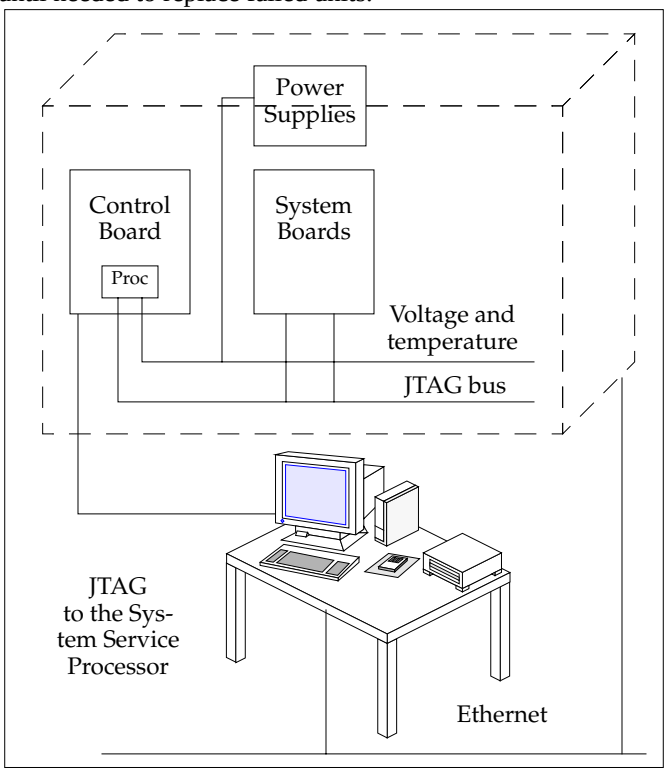

Figure 11: System Service Processor.

5.1 System Service Processor

To improve availability and serviceability, we added a System Service Processor (SSP) which provides intelligent and known-good diagnostic support. The software of the System Service Processor directs the testing and booting of the system, the dynamic reconfiguration and hot swapping of system components, and the diagnosis of hardware concurrently with system operation.

The System Service Processor is a SPARC workstation, configured with local disk, a CD-ROM drive, and an Ethernet. It is connected to the hardware through a special JTAG interface, which is a separate back door path to all of the ASICs in the system. It uses the JTAG path to diagnose and configure the machine, to monitor the health of the system, and to collect error information in the case of a failure.

The System Service Processor runs a set of software which includes the *hostmon* daemon, an open boot executive, kernel debugger, configuration and bring-up executables, and a heartbeat monitor. It uses the Ethernet to interact with the main Solaris system in the CS6400, and optionally, up to two domains, which are self-contained single-board systems (See Section 5.5).

5.2 Hostview

Hostview is a graphical-user interface (GUI) based application that runs on the System Service Processor and provides control over the CS6400. Hostview provides the following actions to the main system, and the domain systems.

- Power the CS6400 host on and off.
- Dynamically reconfigure the boards within the CS6400, logically attaching or detaching them from the operating system, resetting them, and running diagnostics on them.
- Dynamically reconfigure system boards as independent domains, which operate as a separate environment from the main CS6400 system. A domain can carry its own workload and has its own log messages file.
- Configure the system using Power-On Self Test and boot Solaris.
- Start console windows.
- Access log message files.
- Edit the blacklist file to enable or disable hardware components.

Figure 12: shows an example Hostview main screen. The graphic is a top view of the system, which in this case has 12 system boards. Boards 8 and 9 are shown without a line connected them to the centerplane, meaning that they have been isolated into their own domains. Each processor is represented by a colored square, which indicates the state of each processor: running, blacklisted, or not configured. A small icon inside each square shows what is running: a diamond indicates Solaris.

Hostview can be run remotely from any networked X-Windows device, and a command-line version is available for remote operation over dialup lines.

5.3 Power On Self Test

Power-On Self Test (POST) runs under System Service Processor control at start-of-day, reboot, and after a panic. First, each processor checks itself, its cache and its interfaces to the XDBuses. This initial phase is executed from memory located on the BootBus, rather than from main memory, as the XDBus is not yet enabled. Next, the bus interfaces are tested. Finally, the XDBus is turned on and the system is tested as a whole,

starting with memory units and I/O units. Operational units are noted.

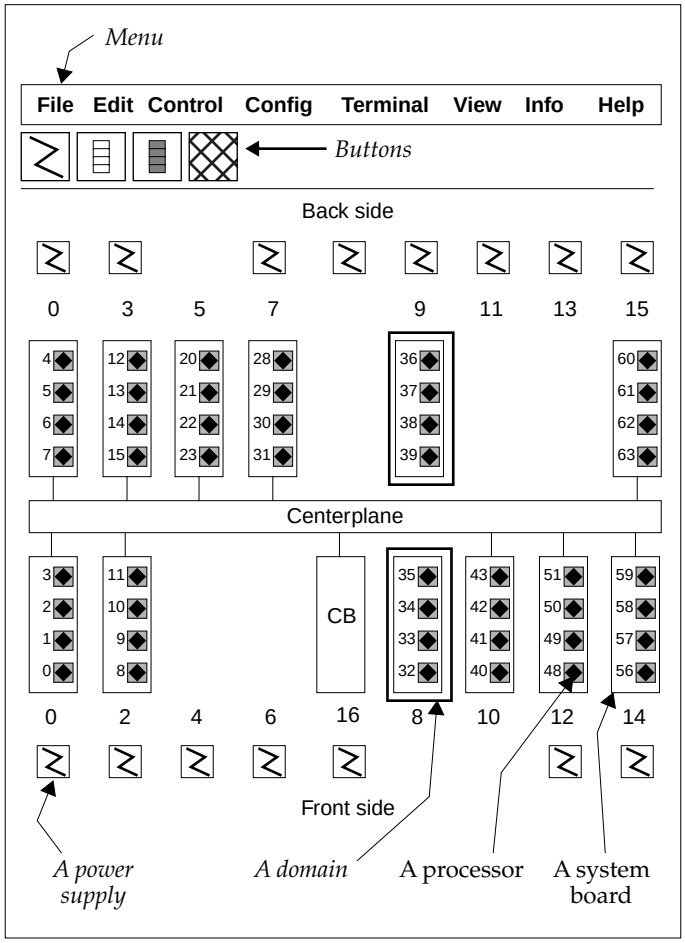

Figure 12: Example Hostview main screen.

The System Service Processor configures the system using only fully-functional components, including assigning addresses to memory banks. Then the system is booted and status is reported by the System Service Processor. Bad components are not logically configured, and may be serviced at a later time.

5.4 Dynamic reconfiguration

On most Unix servers, removing a board for service requires that the system be shut down. We added *dynamic reconfiguration* and *hot swap* to the CS6400, which permits the system to be serviced while remaining operational. Dynamic reconfiguration and hot swap allow a system board to be logically detached from the operating system, then physically removed and then later reinserted, and finally logically reattached — all concurrent with system operation, and without requiring a power-down and reboot.

The dynamic reconfiguration software is a combination of host software and software on the System Service Processor. The System Service Processor communicates using remote procedure calls to a daemon running on the CS6400. This daemon communicates with the CS6400 Solaris kernel. The Hostview interface provides an interactive program to guide the System Administrator or Field Engineer through the logical detach and attach steps.

Dynamic reconfiguration and hotswap is accomplished in five steps, as shown in Figure 13:.

1. **Detach the bad board**. The operator requests that a system board be logically detached. The kernel flushes all the processes, swaps out the user pages, and physically remaps the kernel pages away from the board — which logically detaches the board from the system.

2. **Hot remove the bad board**. The service provider attaches a 5-volt power cord to the board, and pushes a button indicating that he is ready to remove the board. The System Service Processor requests the kernel to quiesce Solaris. All XDBus activity must cease for a few seconds. The operator removes the board, and pushes a button indicating that he is done. The System Service Processor tells the kernel to resume normal Solaris execution.

3. **Hot insert the new board**. The service provider attaches a 5-volt power cord to the board, and pushes a button indicating that she is ready to insert the board. The System Service Processor requests the kernel to quiesce Solaris. All XDBus activity must cease for a few seconds. The operator inserts the board, and pushes a button to indicate she is done. The System Service Processor tells the kernel to resume Solaris.

4. **Debut the new board**. The System Service Processor runs Power On Self Test on the board. This tests the board, configures it, and leaves it enabled on the centerplane. Debuting the board requires Solaris to quiesce for a brief period so that the connection between the board and the rest of the system can be checked to ensure that the rest of the system will not be harmed by the new board.

5. **Attach the new board**. The operator requests that the kernel begin using the inserted board. The kernel adds the new board's resources into the system.

5.5 System domains

System domains are a well-known feature in the mainframe world. They provide the ability to run a separate instance of the operating system on a part of the hardware that is isolated from the rest of the system. For the CS6400, domains are created by isolating a single system board from the rest of the CS6400 system. This board can be used as a completely independent Solaris system.

Domains add to the availability of the CS6400 by creating a separate environment for safely bringing up and testing new software. A domain may be used to test and debug new mission-critical application software, or to break-in a new Solaris release without impacting production users. Because a domain is physically isolated from the XDBus, software and hardware errors inside the domain cannot affect the rest of the

system. Equally, errors in the remaining portion of the system will not affect the domain.

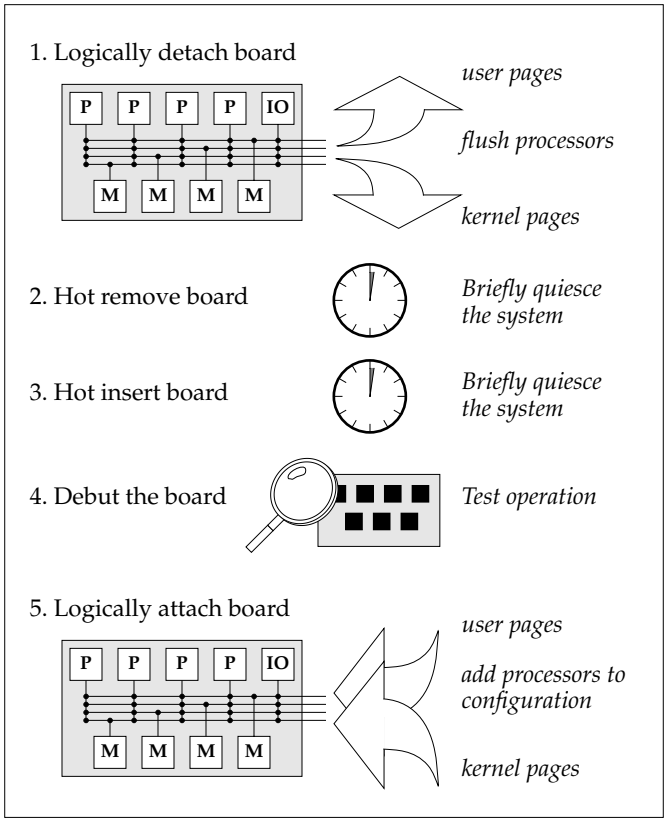

Figure 13: Dynamic reconfiguration.

Because each domain is its own computer system, it must have appropriate peripheral and network connections. Each domain must be configured with a disk to boot from, a network connection, and sufficient memory and disk space to accomplish the intended task. Because a domain is limited to one board, there is a maximum configuration of four processors per domain. There is an overall limit of two domains, in addition to the main CS6400 system.

The System Service Processor acts as the system console for the main instance of Solaris and for each domain. A system board can be dynamically detached from the system to create a new domain. Solaris can then be booted up on the domain, and it is available for immediate use. Each domain has its own host-id, and operates as a completely separate computer system. Dynamic reconfiguration attach may later be used to return the system board used in the domain back to the main CS6400 system.

5.6 Alternate pathing

Alternate pathing provides a mechanism for switching from one SBus I/O adaptor card to another without disturbing the processes that are using the connected I/O devices. The primary goal of alternate pathing is to support dynamic reconfiguration. When a system board is removed, access to the connected devices needs to be switched to use adapters on other system boards, if possible. This allows continued system operation after the system board is removed.

Alternate pathing can allow recovery from failed or failing adapters by allowing different adapters to be used. It also guarantees that a system can boot unattended even if the primary network or boot disk is not available.

The system administrator defines the physical paths (i.e., SBus adapters) that are equivalent, and indicates which path is the primary path. Alternate pathing creates a *meta-device* name for each SCSI adapter path group, and a *meta-network* name for each network adapter path group. This meta-name is used to access one or more physical device or network names, and remains unchanged regardless of which physical path is used. By using the meta-names, switching from one path to another has no impact on the running scripts or applications.

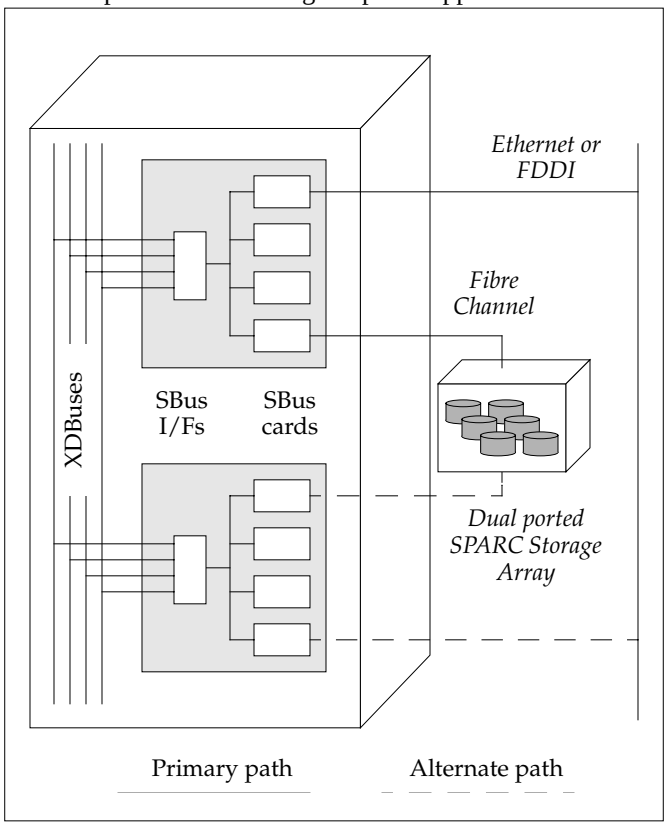

Figure 14: Alternate pathing.

6 UNICOS value-added software

6.1 Cray Tape Management System

Tape support on Unix systems has typically been restricted to basic read/write functionality. Mainframe and supercomputer users expect to request a tape and have it mounted by either an operator or a robotic device. They assume support for standard ANSI and IBM labeled formats is provided at the system level, so that security is enforced for all tape accesses.

We added a Tape Management System (TMS) to Solaris that is modeled after the field-proven UNICOS tape subsystem on Cray Research's parallel/vector supercomputers. Applications can request a tape and have it mounted by either an operator or

a robotics device. Support for standard ANSI and IBM labeled formats is provided at the system level so that security is enforced for all tape accesses. Because the tape drives are managed by the Tape Management System, it is possible to securely share tape drives, including robotic devices, among all users on the system.

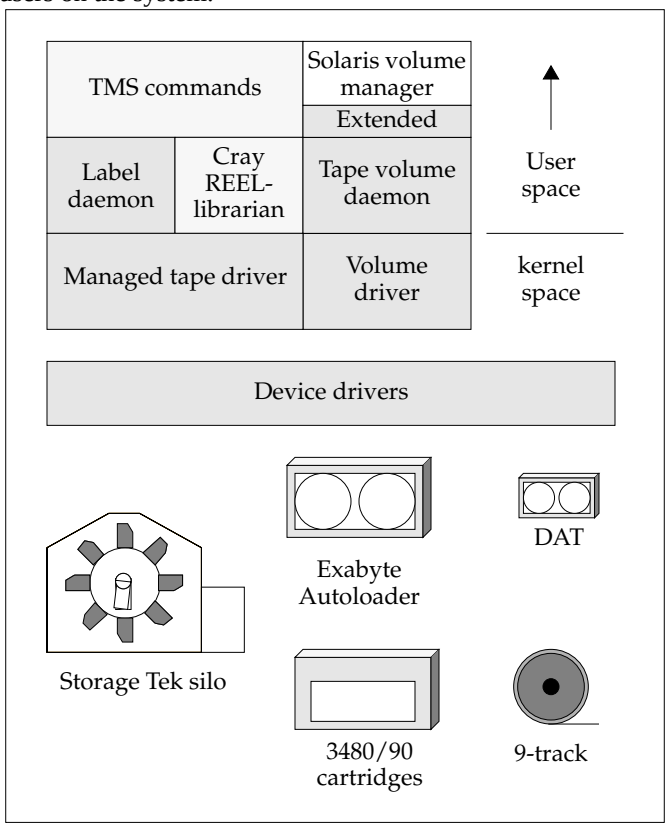

Figure 15: Tape Management System (TMS).

The Tape Management System is comprised of several layered components, as shown in Figure 15::

1. A new version of the Solaris removable-media Volume Manager, with our extensions for tapes, controls the reservation of drives and mounting of volumes.
2. The label daemon supervises all file access transitions, handles tape marks, and processes all labels.
3. The layered device drivers supervise the normal tape I/Os and route exceptions to the appropriate daemons for processing.
4. The Cray REELlibrarian tape catalog.

We used the Solaris's removable media Volume Manager to insure a smooth fit with the Solaris. Our enhancements include support for tape, multi-volume tape, multiple drive capabilities, label processing (specifically IBM and ANSI labels), and an interface to Cray REELlibrarian tape catalog.

The Cray REELlibrarian communicates with the Tape Management System to provide a secure, easy-to-use volume management system combining a complete on-line catalog of all volume and file information with the management and control of tape access.

6.2 LibSci numerical library

LibSci is the numerical library supported on all Cray Research products. It provides over 1,200 routines that are used for scientific and technical computing, such as solving systems of linear equations, computing eigenvalue and eigenvectors, manipulating matrices, solving sparse equations, and filtering signals and images.

The library includes the industry-standard Basic Linear Algebra Subroutines (BLAS), Linear Algebra Package (LAPACK), as well as 1D, 2D, and 3D FFT and convolution routines, random number generators, and matrix transposition routines. The LibSci routines are callable from Fortran 77, Fortran 90, and C. CraySoft sells this package for the complete range of Solaris/SPARC platforms.

We had to re-implement the algorithms in LibSci to match the characteristics of a cache-based shared-memory architecture.

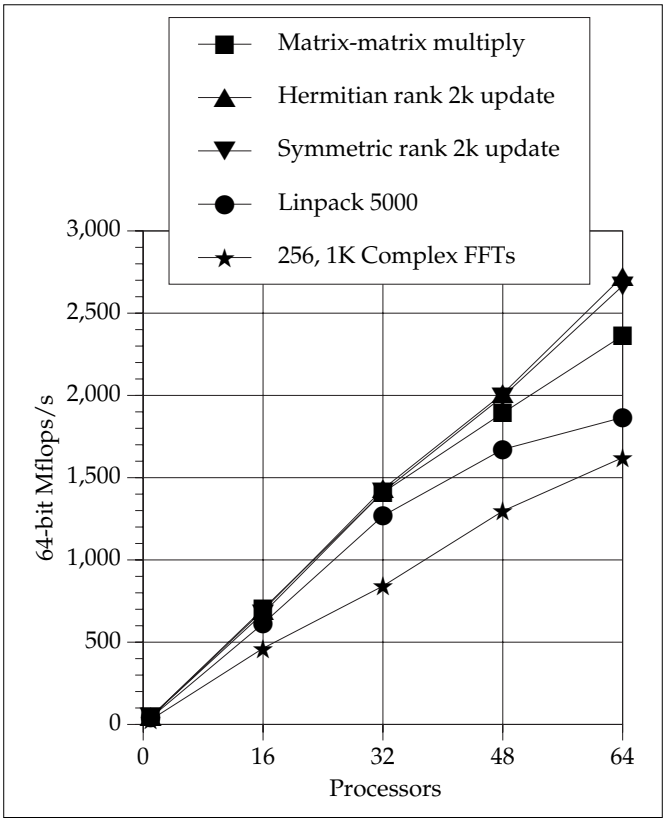

Figure 16: LibSci performance on the CS6400.

Dual precisions. CS6400 systems support both 32-bit single precision, and 64-bit double precision. Thus we had to double the number of routines over the parallel/vector version of LibSci, which works only in 64-bit precision.

Fast synchronization. Quick synchronization is necessary to allow relatively small chunks of work to be done in parallel. Solaris provides a set of synchronization primitives in its thread

library, but they are designed for a coarser grain of parallelism. As a result, its synchronization times were too slow.

We implemented our own ultra-light weight thread synchronization routines which spin-lock — rather than return to the operating system — when the threads are closely aligned and the wait times are very short. We implemented a butterfly-type barrier using shared memory, whose execution time rises only logarithmically to 28 microseconds for 64 processors.

Cache blocking. The caches of the SuperSPARC comprise a 128 Mbytes scratch space which the processor can access at an aggregate bandwidth of over 28 Gbytes per second. This is 25 times the rate to memory. Reusing cached data is the key to high performance on the CS6400, as it is on all microprocessor-based systems. We organized the inner loops of the matrix and FFT routines to reuse data that is already in the cache as much possible. Typically this means working on groups of vectors at a time, before moving on to the next set of data.

Blocking is also the key to avoiding strided memory references. On the CS6400, data moves to and from memory in 64-byte chunks. Our LibSci algorithms are optimized to take advantage of the CS6400's block memory accesses to avoid wasteful memory access patterns. For example, matrix transposition is done by breaking up the matrix into 64 byte by 64-byte tiles that make use of every byte in a cache line, rather than by reading a contiguous column and scattering it out to a row as one would do on a vector memory system.

The result of our work is an implementation of LibSci that is optimized to exploit cached, shared memory, multiprocessing architectures. Figure 16: shows the multi-processor performance we obtained for several 64-bit precision LibSci routines.

7 Summary

SunSoft has made Solaris the environment of choice for *open* enterprise computing. SunSoft's top priority has been extending the reliability of the Solaris environment. Their other priorities for new releases include compatibility with previous releases, increasing performance, and adding features for large servers.

Our most important task is to guarantee compatibility between Solaris on the CS6400 and on Sun Microsystems platforms. We also work to improve our performance scaling, to enhance our reliability, availability, and serviceability features, and to add new layered features to Solaris that have been proven in UNICOS to be needed for large systems.

The Cray / Sun partnership has born fruit with the most powerful symmetric multiprocessing system on the market. The CS6400 has proven capable of tackling problems that were previously solvable only on mainframes. We look forward to developing ever more powerful superservers based on faster SPARC processors, higher bandwidth interconnects, and the Solaris networked computing environment.