An Oracle White Paper
April 2011

Performance Considerations for Developers Utilizing Oracle's SPARC Enterprise M-Series Servers

ORACLE

Introduction

Oracle's SPARC Enterprise M-Series server product family offers a scalable range of compute power, from Oracle's SPARC Enterprise M3000, M4000, and M5000 servers in rackmount form factor to Oracle's SPARC Enterprise M8000 and M9000 servers in datacenter cabinets (Figure 1). These systems implement a common high-performance architecture and utilize the latest SPARC64® processors, helping organizations maximize solution performance.

Figure 1. SPARC Enterprise M-Series Servers

This technical white paper discusses the characteristics of SPARC64 processors, attributes of each individual SPARC Enterprise M-Series server, and I/O subsystem behavior. Within this document, an emphasis is placed on how system features impact application performance. In addition, the article provides some insights with regard to gaining the best performance from SPARC Enterprise M-Series servers and large computer systems in general. The content of this article assumes a basic understanding of server architecture, multithreaded processors, virtualization technology, and performance tuning principles.

The document addresses the following topics:

- "Characteristics of SPARC64® VI and SPARC64 VII/VII+ Processors", discusses the architectural design and features of SPARC64 VI and SPARC64 VII/VII+ processors.

- "Characteristics of SPARC Enterprise M-Series Servers", provides a detailed review of the system interconnect, memory subsystem, and I/O subsystem of SPARC Enterprise M-Series servers, as well as information regarding migrating from existing Sun platforms to SPARC Enterprise M-Series servers.

- "Oracle Solaris Improvements for SPARC Enterprise M-Series Servers", highlights specific features of the Oracle Solaris operating system that aim at improving the performance of applications that execute on SPARC Enterprise M-Series servers.

- "Advanced Options for Maximizing Performance", describes how utilizing the latest compilers and the scheduling optimization capabilities provided by Oracle Solaris can help improve application performance.

- "System Tools for Understanding System Behavior", briefly highlights a few tools for monitoring system performance.

- "Summary", summarizes the main points presented in this paper.

- "For More Information", includes links to references.

Characteristics of SPARC64® VI and SPARC64 VII/VII+ Processors

SPARC64 VI and SPARC64 VII processors maintain many similarities along with a few significant differences. The instruction processing for both processor models is 4-way super-scaler, allowing up to four instructions to be issued per cycle per core. In addition, the cores support out-of-order execution, providing the ability for multiple instructions to fetch data from memory at the same time. For both processors, two strands — also known as active thread contexts — share each core. When both strands are active, throughput is maximized. When only one strand is active, response time for the thread running on that strand is optimized.

The primary difference between the SPARC64 VI and SPARC64 VII/VII+ processor lies in the number of cores supported by each processor and the approach to multithreading. More information on the differences between SPARC64 VI and SPARC64 VII/VII+ processors can be found in the section titled "Multithreaded Cores". Table 1 highlights the primary characteristics of the SPARC64 VI and SPARC64 VII/VII+ processors.

TABLE 1. COMPARISON OF SPARC64 VI AND SPARC64 VII/VII+ PROCESSORS

	SPARC VI PROCESSOR	SPARC VII PROCESSOR	SPARC VII+ PROCESSOR
SPEED	• 2.15 GHz • 2.28 GHz • 2.4 GHz	• 2.4 GHz • 2.52 GHz • 2.53 GHz • 2.77 GHz • 2.88 GHz	• 2.66 GHz • 2.86 GHz • 3.0 GHz
ARCHITECTURE	• Dual-core • SPARC V9 • sun4u • 90 nm process technology	• Quad-core • SPARC V9 • sun4u • 65 nm process technology	• Quad-core • SPARC V9 • sun4u • 65 nm process technology
L1 CACHE	• 128 KB L1 I-cache per core • 128 KB L1 D-cache per core	• 64 KB L1 I-cache per core • 64 KB L1 D-cache per core	• 64 KB L1 I-cache per core • 64 KB L1 D-cache per core
L2 CACHE	• 5 MB (2.15 GHz and 2.28 GHz) • 6 MB (2.4 GHz) • 10-way associative (2.15 and 2.28 GHz) • 12-way associative (2.4 GHz) • 256 byte line size • ECC tag and data protection	• 5 MB (2.4 GHz and 2.77 GHz) • 5.5 MB (2.53 GHz) • 6 MB (2.52 GHz and 2.88 GHz) • 12-way associative • 256 byte line size • ECC tag and data protection	• 5.5 MB (2.86 GHz) • 11 MB (2.66 GHz) • 12 MB (3.0 GHz) • 12-way associative • 256 byte line size • ECC tag and data protection
POWER	• 120 watts nominal • 150 watts maximum	• 135 watts nominal • 150 watts maximum	• 135 watts nominal • 150 watts maximum

Instruction Compatibility and Extensions

Both processors provide full SPARC V7, V8, and V9 binary compatibility. With the proper compiler settings, both processors also support the fused floating-point multiply-add instruction group which can double the peak floating-point execution rate. With the fused floating-point multiply-add, two multiply-add operations can be initiated every cycle, for a peak rate of four floating-point operations per processor cycle per core. In addition to the floating-point extensions, the processors include `SLEEP` and `SUSPEND` privileged instructions for controlling the dual-strand nature of the processor.

TLB Structure

The SPARC64 VI and SPARC64 VII/VII+ processors offer the same TLB structure. Page sizes of 8 KB, 64 KB, 512 KB, 4 MB, 32 MB, and 256 MB are supported. A complete set of identical TLB registers exist in system space and user space. For each space, there is a small fully-associative tlb (ftlb) with 32 entries that support any or all page sizes at the same time. There are two large set associative tlbs (stlb) with 1024 entries each that support one page size at a time. If the two stlbs are set to use the same page size, then they function as one 2048-entry stlb.

Multithreaded Cores

In both processor types, each hardware strand is presented to the Oracle Solaris operating system as a processor. For example, the `mpstat(1)` and `psradm(1M)` commands list four processors for each dual-core SPARC64 VI chip and eight processors for each quad-core SPARC64 VII/VII+ chip. As provided by the following descriptions, SPARC64 VI and SPARC64 VII/VII+ processors take slightly different approaches to multithreading support.

- **SPARC64 VI processor multithreading**

  For SPARC64 VI processors, either one strand or the other "owns" the instruction pipeline at any given time. As a result, a context switch is required to change the active strand. This approach is known as vertical multithreading (VMT). The context switch is controlled entirely in hardware and is invisible to the operating system. A context switch is triggered either by an L2 cache miss or when 3000 cycles have passed without a context switch. The context switch takes 21 cycles. Since an L2 cache miss can take several hundreds of cycles, having two strands allows the continuation of useful work instead of setting the processor completely idle while waiting for memory to respond.

  Throughout operation of the SPARC64 VI processor there is no attempt to provide fairness between the two threads. If the two threads have roughly the same L2 cache miss rates, each receives roughly the same share of the processing time. If one thread takes many L2 cache misses while the other seldom misses the L2 cache, the thread which seldom misses the L2 cache gains a much larger share of the processing time.

- **SPARC64 VII/VII+ processor multithreading**

  The SPARC64 VII/VII+ processor provides an improved multi-strand model. Both strands share the instruction pipeline, alternating instruction initiation every cycle. This approach is known as

Simultaneous Multithreading (SMT). On the first cycle, the first strand issues a group of instructions. On the second cycle, the second strand issues a group of instructions — then back to the first strand, and the cycle continues. Instructions from either strand progress through the instruction pipeline according to functional unit and input data availability. There are two integer, two floating point, two load/store, and one branch functional unit available to execute instructions. Any functional unit can handle instructions from either thread. This flexible design allows both threads to gain roughly equal access to the processor. Throughput results depend on the application characteristics. Utilizing two threads can result in very small throughput gains if both threads are competing for a single resource. In contrast, if the two strands are limited by other factors such as memory latency, utilizing the SMT processing model can lead to nearly double the effective throughput.

For both the SPARC64 VI and SPARC64 VII/VII+ processor models, the `SUSPEND` and `SLEEP` instructions provide execution control. The `SUSPEND` instruction causes a strand to stop executing instructions until an interrupt is received telling the strand to start again. This instruction is used when Oracle Solaris has been told to deactivate a processor — as with execution of the command `psradm -f`. The `SLEEP` instruction causes a strand to stop executing instructions for 1.5 microseconds, potentially improving the throughput of the other strand. This instruction is used by Oracle Solaris when the processor is in the idle loop, waiting for work. Measurements have shown that putting a strand in the `SLEEP` idle loop is similar to having the strand in a `SUSPEND` state, to within 2% overall throughput.

Processor to Memory Interface

Within the SPARC64 VI and SPARC64 VII/VII+ processors, all cores share the L2 cache and the interface from the processor to the memory subsystem. Data is moved from memory to the processor either when a cache miss occurs or when a prefetch is issued. Prefetches can be either hardware or software derived. In all cases, if the data is not in the cache of another processor, then a 256 byte cache line is moved into the processor. If the data is in the cache of another processor, only a 64-byte sub-block of the cache line is moved. This measure reduces false-cache thrashing.

When data is obtained from the cache of another processor, the data first moves from the other processor's cache to memory and then to the cache of the requesting processor. As a result, the latency for the transfer is double that of a memory to processor transfer. Using the block-load or block-store instructions only moves 64 bytes, aligned on a sub-block boundary and does not affect the cache. There can be 16 read and 16 writes in flight per chip at any point.

Prefetching

The SPARC64 VI and SPARC64 VII/VII+ processors support both hardware-initiated and software-initiated prefetching. Hardware prefetching is initiated as follows. Whenever there is an L2 miss for a cache line, the cache line address is placed in the 16-entry prefetch queue (PFQ). If a later L2 miss occurs for the next cache line, then a prefetch is issued for the following cache line. When several prefetches have been triggered, the second cache line following is also prefetched. These prefetches occur without any software intervention and typically reduce memory latency for regular data accesses.

Hardware prefetches are low priority and are canceled if enough other memory requests occur to fill the request queue.

Software prefetches can be used to explicitly request data to be fetched into either the L2 cache or both the L1 data cache and L2 cache. The prefetch instruction supports a "one" or "many" option. If the "one" option is used, then the data is only placed in the L2 cache. If the "many" option is used, the data is placed in the L1 data cache and L2 cache. These prefetches also specify whether the cache line is to be prefetched for reading or writing.

Tuning software prefetches for maximum performance requires some understanding of a system's memory latency and the behavior of prefetches. Software prefetches are lower priority than direct memory references. When the read or write queue (as appropriate) is full, additional software prefetches are dropped. The performance tuning implications of this behavior are discussed in the section titled "Advanced Options for Maximizing Performance".

Characteristics of SPARC Enterprise M-Series Servers

Oracle's SPARC Enterprise M-Series server family offers six system models, including Oracle's SPARC Enterprise M3000, M4000, M5000, M8000, M9000-32, and M9000-64 servers. Each system supports up to eight memory DIMMs per processor chip. Current DIMM size offerings include 1 GB, 2 GB, 4 GB, and 8 GB densities. SPARC Enterprise M3000, M4000, M5000 servers are rackmount units while the SPARC Enterprise M8000 and M9000 servers offer a datacenter cabinet form factor. Table 2 provides a comparison of the high level characteristics of these systems.

TABLE 2. SPARC ENTERPRISE M-SERIES SERVER CHARACTERISTICS

	SPARC ENTERPRISE M3000 SERVER	SPARC ENTERPRISE M4000 SERVER	SPARC ENTERPRISE M5000 SERVER
ENCLOSURE	2 rack units	6 rack units	10 rack units
SPARC64 VI PROCESSORS	• N/A	• 2.15 GHz • 5 MB L2 cache • Up to four dual-core chips	• 2.15 GHz • 5 MB L2 cache • Up to eight dual-core chips
SPARC64 VII PROCESSORS	• 2.52 GHz and 2.77 GHz • 5 MB L2 cache • Four-core	• 2.4 GHz with 5 MB L2 cache • 2.53 GHz with 5.5 MB L2 cache • Up to four quad-core chips	• 2.4 GHz with 5 MB L2 cache • 2.53 GHz with 5.5 MB L2 cache • Up to eight quad-core chips
SPARC64 VII+ PROCESSORS	• 2.86 GHz • 5.5 MB L2 cache • Four-core	• 2.66 GHz with 11 MB L2 cache • Up to four quad-core chips	• 2.66 GHz with 11 MB L2 cache • Up to eight quad-core chips
MEMORY	• Up to 64 GB • Eight DIMM slots	• Up to 256 GB • 32 DIMM slots	• Up to 512 GB • 64 DIMM slots
INTERNAL I/O SLOTS	• Four PCI Express	• Four PCI Express • One PCI eXtended	• Eight PCI Express • Two PCI eXtended
EXTERNAL I/O CHASSIS	One external x2 SAS port	Up to two units	Up to four units
INTERNAL STORAGE	• Serial Attached SCSI • Up to four drives	• Serial Attached SCSI • Up to two drives	• Serial Attached SCSI • Up to four drives
DYNAMIC DOMAINS	One	Up to two	Up to four

	SPARC ENTERPRISE M8000 SERVER	SPARC ENTERPRISE M9000-32 SERVER	SPARC ENTERPRISE M9000-64 SERVER
ENCLOSURE	One cabinet	One cabinet	Two cabinets
SPARC64 VI PROCESSORS	• 2.28 GHz with 5 MB L2 Cache • 2.4 GHz with 6 MB L2 cache • Up to 16 dual-core chips	• 2.28 GHz with 5 MB L2 cache • 2.4 GHz with 6 MB L2 cache • Up to 32 dual-core chips	• 2.28 GHz with 5 MB L2 cache • 2.4 GHz with 6 MB L2 cache • Up to 64 dual-core chips
SPARC64 VII PROCESSORS	• 2.52 GHz and 2.88 GHz • 6 MB L2 cache • Up to 16 quad-core chips	• 2.52 GHz and 2.88 GHz • 6 MB L2 cache • Up to 32 quad-core chips	• 2.52 GHz and 2.88 GHz • 6 MB L2 cache • Up to 64 quad-core chips
SPARC64 VII+ PROCESSORS	• 3.0 GHz with 12 MB L2 cache • Up to 16 quad-core chips	• 3.0 GHz with 12 MB L2 cache • Up to 32 quad-core chips	• 3.0 GHz with 12 MB L2 cache • Up to 64 quad-core chips
MEMORY	• Up to 1 TB • 128 DIMM slots	• Up to 2 TB • 256 DIMM slots	• Up to 4 TB • 512 DIMM slots
INTERNAL I/O SLOTS	32 PCI Express	64 PCI Express	128 PCI Express
EXTERNAL I/O CHASSIS	Up to 8 units	Up to 16 units	Up to 16 units
INTERNAL STORAGE	• Serial Attached SCSI • Up to 16 drives	• Serial Attached SCSI • Up to 32 drives	• Serial Attached SCSI • Up to 64 drives
DYNAMIC DOMAINS	Up to 16	Up to 24	Up to 24

Memory Interconnect

Within the SPARC Enterprise M-Series server family, the memory interconnect structure varies based on the size of the system. For all SPARC Enterprise M-Series servers with more than one processor, the building block is a system board containing up to four processor chips. Memory is placed equally distant from each processor on a given system board. Off-board memory — memory on another system board within the same server — is further than memory that is on the same board as a given processor. For these reasons, Oracle Solaris considers all processors and memory within a system board to be in a common locality group for scheduling purposes. Additional information on locality groups is found in the section titled "Oracle Solaris Improvements for SPARC Enterprise M-Series Servers".

System-wide cache coherency is maintained by the system control (SC) chips. Every reference to memory passes from a processor through one or more SC chips. The SC chips then interface to memory access control (MAC) chips. Oracle's SPARC Enterprise M3000 server is an exception to this general rule. Since this server offers a single-processor design, the SC is simplified and combined with the MAC to form a Jupiter System Controller (JSC) chip (Figure 2). This design offers higher bandwidth and lower latency than the more general solutions that multiprocessor systems require. Oracle's SPARC Enterprise M4000 and M5000 servers support two processors per SC chip. This approach saves space and cost at the expense of bandwidth. In contrast, Oracle's SPARC Enterprise

M8000 and M9000 servers assign one processor per SC chip, optimizing potential bandwidth to meet the needs of larger systems.

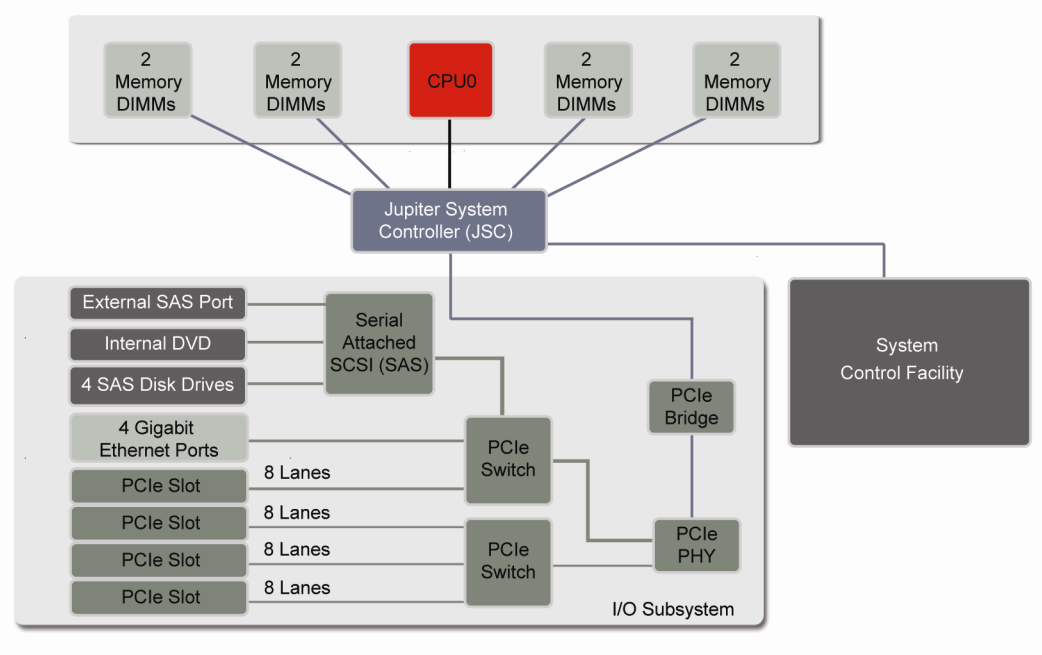

Figure 2. SPARC Enterprise M3000 server architecture diagram

The SPARC Enterprise M4000 server consists of a single system board. As shown in Figure 3, each SC chip connects to all four processors and controls cache coherency for half of the memory chips. As a result, all memory provides the same latency. The SPARC Enterprise M5000 server contains two system boards. Looking at Figure 4, the two SC chips on each board connect to the four processors on that board and to the SC chips on the other board. When a processor accesses memory on the other board, an additional "hop" is required and induces a higher latency. In addition, memory accesses on SPARC Enterprise M5000 servers have a slightly higher latency than the M4000. This effect results from the cache coherency protocol requiring communication to the farthest SC chip before resolving a memory reference.

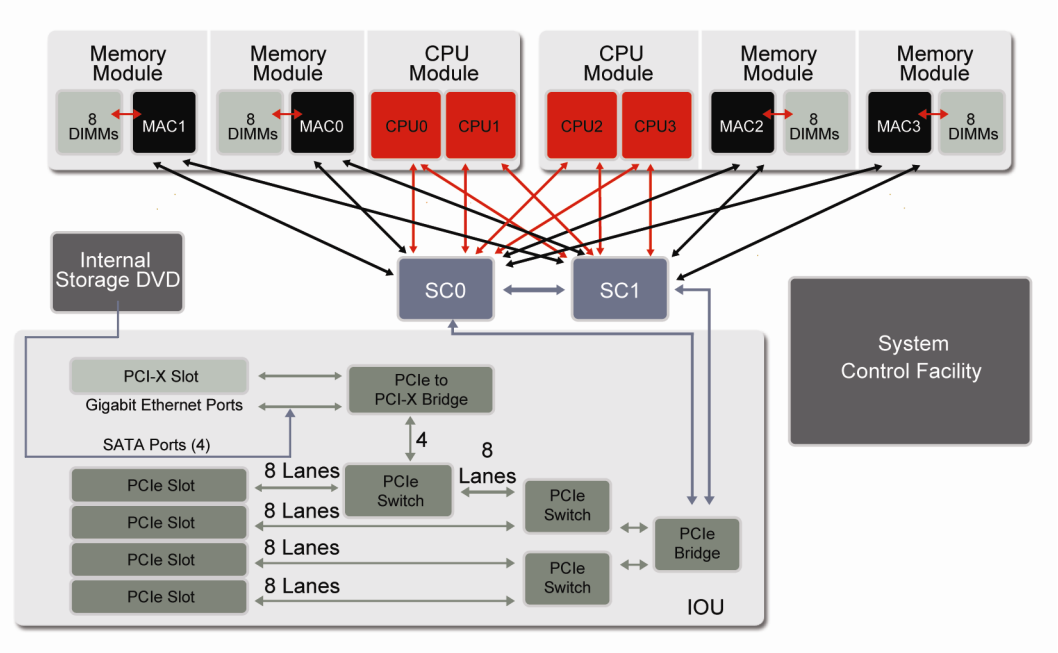

Figure 3. SPARC Enterprise M4000 server architecture diagram

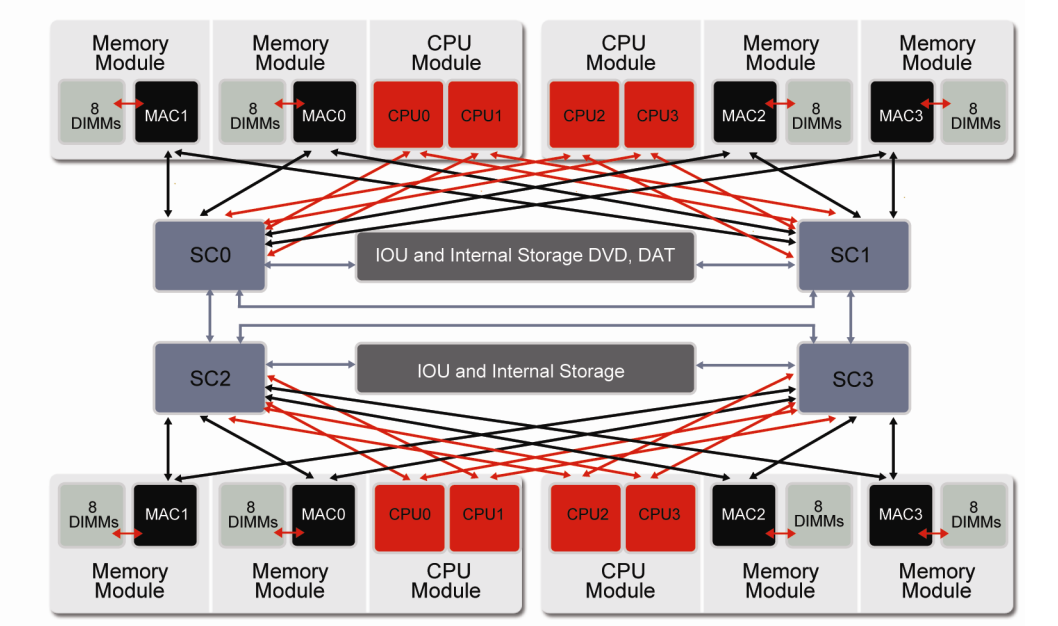

Figure 4. SPARC Enterprise M5000 server architecture diagram

SPARC Enterprise M8000 and M9000 servers require larger building blocks to support more scalable configurations. The SPARC Enterprise M8000 server provides the basic structure and the M9000 builds on that structure. The basic architecture for these servers is shown in Figure 5.

Within the SPARC Enterprise M8000 server, each system board contains four SC chips and four processors. SPARC Enterprise M8000 servers can contain up to four system boards. Each SC chip directly connects to each processor. Latency for all memory on the board is equal for all processors. Each SC chip connects to the crossbar (XB) board to provide communication between the four boards in the SPARC Enterprise M8000 server. Memory accesses that go through the XB board are all equal in latency to each other and higher in latency than memory accesses to the local board for the processor making the access.

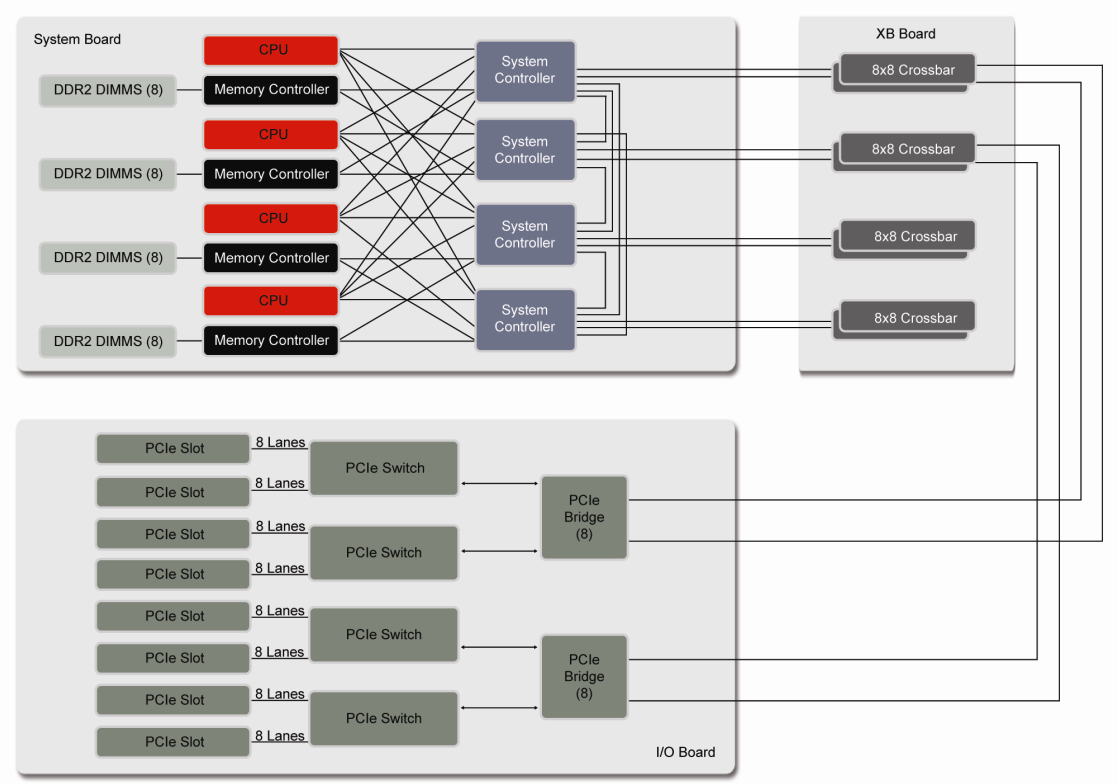

Figure 5. SPARC Enterprise M8000 and M9000 server system and I/O board diagram

The SPARC Enterprise M9000-32 server can be thought of as two SPARC Enterprise M8000 systems with an additional connection between the XB boards. Memory accesses within a board provide the lowest latency. Accesses within a set of four boards connected by one XB offer the next highest latency. A higher latency is induced by memory accesses, which cross to the other XB. The SPARC Enterprise M9000-64 server carries this principle one step farther, by adding a second cabinet and additional connections. Because of the additional physical distance the signals must travel, accesses that cross between cabinets have a higher latency than data transfers in the same cabinet.

Memory Latency

As described, latency increases with the distance from a processor to memory. In larger systems, even the latency for local accesses increases due to the need to maintain cache coherence across the entire server. In addition, systems with SPARC64 VII/VII+ processors exhibit a higher latency of 8 to 12

nanoseconds due to the need to arbitrate between four processor cores within each chip instead of only two cores as on the SPARC64 VI processor.

Table 3 shows the memory latency of an unloaded SPARC Enterprise server with SPARC64 VI processors. The column labeled "nearest" provides the latency for memory access on the same system board. The numbers in the column labeled "farthest" represent the latency for access to memory that is placed on a system board that is the farthest away. The ratio provides a comparison of the nearest to the farthest. When data is in the cache of another processor, the latency doubles.

TABLE 3. MEMORY LATENCY FOR SPARC ENTERPRISE M-SERIES SERVERS WITH SPARC64 VI PROCESSORS

	NEAREST	FARTHEST	RATIO
SPARC ENTERPRISE M3000 SERVER	220 ns	220 ns	1
SPARC ENTERPRISE M4000 SERVER	245 ns	245 ns	1
SPARC ENTERPRISE M5000 SERVER	312 ns	353 ns	1.13
SPARC ENTERPRISE M8000 SERVER	342 ns	402 ns	1.18
SPARC ENTERPRISE M9000/32 SERVER	387 ns	464 ns	1.20
SPARC ENTERPRISE M9000/64 SERVER	437 ns	532 ns	1.22

Memory Bandwidth

Another important dimension of basic system performance is memory bandwidth. For SPARC Enterprise M-Series servers, there are several factors that affect the achievable bandwidth. For example, the availability and use of optimized software prefetching can help improve bandwidth. In contrast, the following conditions can result in considerably lower bandwidth:

- Memory accesses are random and prefetching is not used
- Data being read by a processor is fetched in 64 byte chunks due to the use of block-load and block-store instructions
- Data being read by a processor is fetched in 64 byte chunks due to the data being in the L2 cache of another processor

There is an absolute limit of one cache line transaction per SC cycle across a domain. For the SPARC Enterprise M4000 and M5000 servers, the SC cycle rate is 1016 MHz. Within the SPARC Enterprise M8000 and M9000 servers, the SC cycle rate is 960 MHz.

The Stream benchmark is a publicly available benchmark that measures system bandwidth (*http://www.streambench.org*). Within the Stream benchmark:

- The Copy operation is a simple array assignment, sufficiently large to well exceed available cache sizes.
- The Triad operation reads two arrays, multiplies one by a scalar, adds the other and then stores the result in a third array so that there are two reads for each write.
- The processor count refers to the number of processor chips in the system.
- The BW/Processor shows the portion of the total system bandwidth that each processor receives.

Table 4 lists the maximum bandwidth observed for each of the SPARC Enterprise M-Series servers with the Stream benchmark while using 256-byte transfers.

TABLE 4. STREAM BENCHMARK RESULTS FOR SPARC ENTERPRISE M-SERIES SERVERS

	THEORETICAL PEAK SYSTEM BANDWIDTH (GB/SECOND)[1]	SNOOP BANDWIDTH (GB/SECOND)	STREAM BENCHMARK TRIAD RESULTS (GB/SECOND)	STREAM BENCHMARK COPY RESULTS (GB/SECOND)	THEORETICAL PEAK I/O BANDWIDTH (GB/SECOND)[2]
SPARC ENTERPRISE M3000 SERVER	22	N/A	5.5	6.6	4
SPARC ENTERPRISE M4000 SERVER	32	129	12.7	12.5	4
SPARC ENTERPRISE M5000 SERVER	64	129	25.2	24.8	8
SPARC ENTERPRISE M8000 SERVER	184	245	69.6	60.3	33
SPARC ENTERPRISE M9000-32 SERVER	368	245	134.4	114.9	61
SPARC ENTERPRISE M9000-64 SERVER	737	245	227.1	224.4	61

Important architectural points to note regarding the bandwidth of SPARC Enterprise M-Series servers include:

- The single-chip design and special JAC chip help the SPARC Enterprise M3000 server provide excellent bandwidth. In fact, the JAC chip can transfer data to a single processor faster than the SC chip.
- The bandwidth of SPARC Enterprise M4000 and M5000 servers is lower than SPARC Enterprise M8000 and larger servers. This lower bandwidth results from the fact that these midrange servers support two processors per SC chip while the larger systems assign one processor per SC chip.
- The SC chip limits throughput to no more than 960 mega-transactions per second per domain. This characteristic sets the theoretical upper bound on bandwidth to 246 GBytes/second. As a result, the SPARC Enterprise M9000-64 server shows a slight reduction in bandwidth per chip. The observed value is within 8% of the theoretical limit, an expected result when allowances are made for occasional contention caused by several processors trying to access the same memory bank.

SPARC64 VI vs. SPARC64 VII/VII+ Processor Scalability

The SPARC64 VII/VII+ processor incorporates twice as many cores as the SPARC64 VI processor, and offers a better multithreading architecture. However, for many applications and benchmarks, the performance gain from utilizing SPARC64 VII/VII+ processors can be notably less than a factor of two. Understanding the common reasons for these limitations can assist in capacity planning and diagnosing system performance limits.

The primary issues that throttle performance relate to cache size and memory bandwidth per chip. The total cache available on the SPARC64 VI and SPARC64 VII processor is identical. The SPARC64 VII+ has double the cache from the SPARC64 VII. Since the SPARC64 VII has double the cores of the SPARC64 VI, the cache available per core is one-half as much. The SPARC64 VII+'s larger cache restores the cache per core to the ratio found in the SPARC64 VI. For applications that require only a small amount of cache, that difference makes little impact. In other cases, there are greater consequences. Applications that utilize all of the cache on the SPARC64 VI processor have less cache per core on the SPARC64 VII processor. As a result, the cache miss rate and need for memory bandwidth increases. Those applications that may have seen limited gains when moving from SPARC64 VI to SPARC64 VII will see larger gains moving from SPARC64 VII to SPARC64 VII+.

Similarly, the total per-chip bandwidth to memory is the same for the SPARC64 VI and VII/VII+ processors. This fact implies that the average per-core bandwidth on the SPARC64 VII/VII+ processor is half of the average per-core bandwidth of the SPARC64 VI processor. Applications with bandwidth requirements well below the available bandwidth are generally not affected. At the other end of the spectrum, applications with high bandwidth demands can fail to realize the full potential of the additional cores. Since the SPARC Enterprise M4000 and M5000 servers provide less bandwidth per core — due to sharing each SC chip between two processors — these systems are more likely to exhibit bandwidth challenges. Some applications can realize a full doubling of performance while others realize smaller gains. Actual results depend on the application profile. Over a broad range of applications, the average gain for doubling the number of cores is around 45% when utilizing SPARC Enterprise M4000 and M5000 servers and 60% when taking advantage of SPARC Enterprise M8000 and M9000 servers.

I/O Subsystems

SPARC Enterprise M-Series server configurations support a maximum of one I/O board for each system board. The design of the I/O boards for SPARC Enterprise M4000 and M5000 servers is different from that of the SPARC Enterprise M8000 and M9000 servers. In all cases, an Oberon bridge chip is used to interface the PCI Express buses to the rest of the system. The maximum I/O transfer size is 64 bytes, a constraint that can be a limiting factor when compared to the 256 byte transfer rate between processors and memory. Splitting a large system into two domains can sometimes increase the total available I/O bandwidth.

The SPARC Enterprise M3000 server provides four PCI Express slots. One PCI Express slot shares a channel with four on-board Gigabit Ethernet connections and a SAS disk controller. The other three PCI Express slots share a different channel. The two channels connect to the Oberon bridge chip, which connects to the JSC chip. An illustration of Oracle's SPARC Enterprise M3000 I/O subsystem is contained in Figure 6.

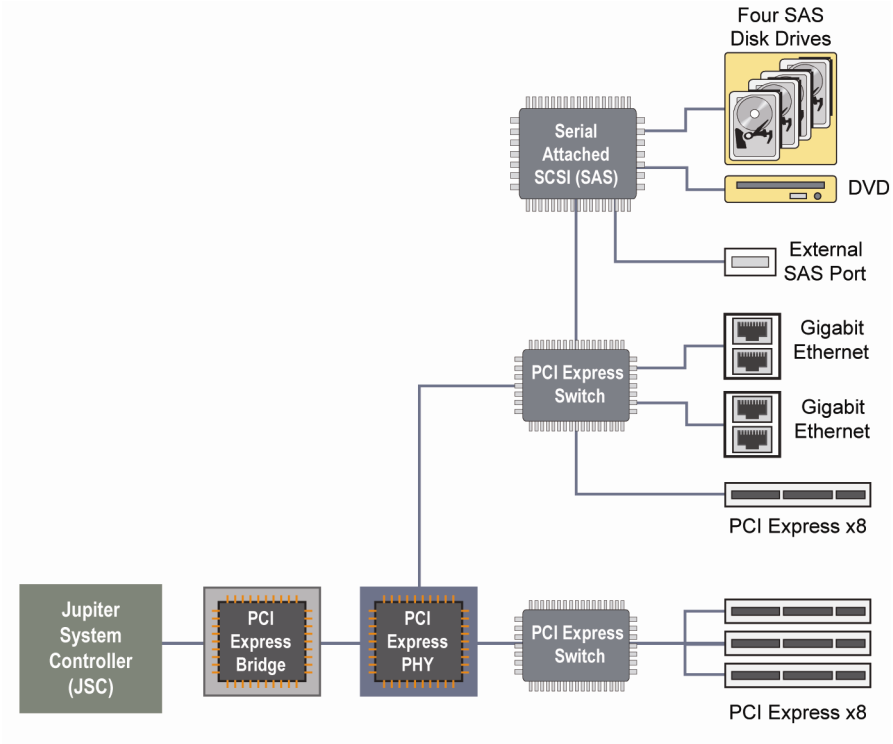

Figure 6. Oracle's SPARC Enterprise M3000 server I/O subsystem

The I/O board on SPARC Enterprise M4000 and M5000 servers supports one PCI-X slot and up to four PCI Express slots (Figure 7 and Figure 8). The PCI-X slot (slot 0) and the first PCI Express slot (slot 1) feed into a PCI Express switch. Slots 0, 1, and 2 combine into one I/O channel and slots 3 and 4 combine into another I/O channel. Both I/O channels feed into an Oberon bridge chip which connects to the SC chips on the system board.

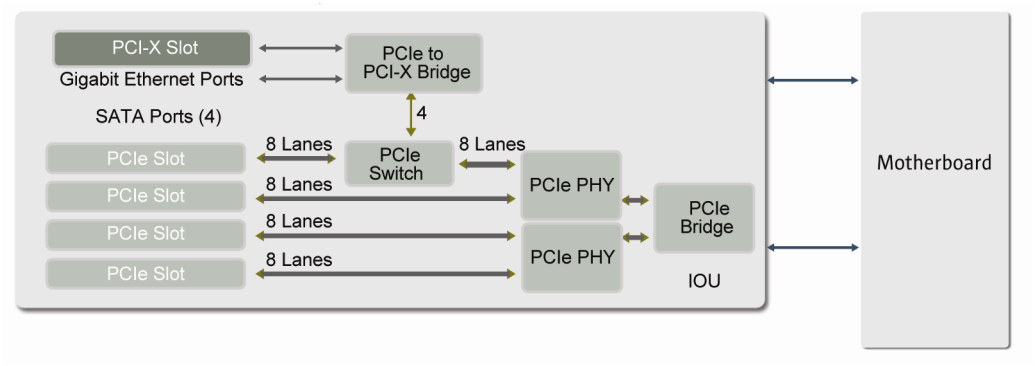

Figure 7. SPARC Enterprise M4000 server I/O subsystem

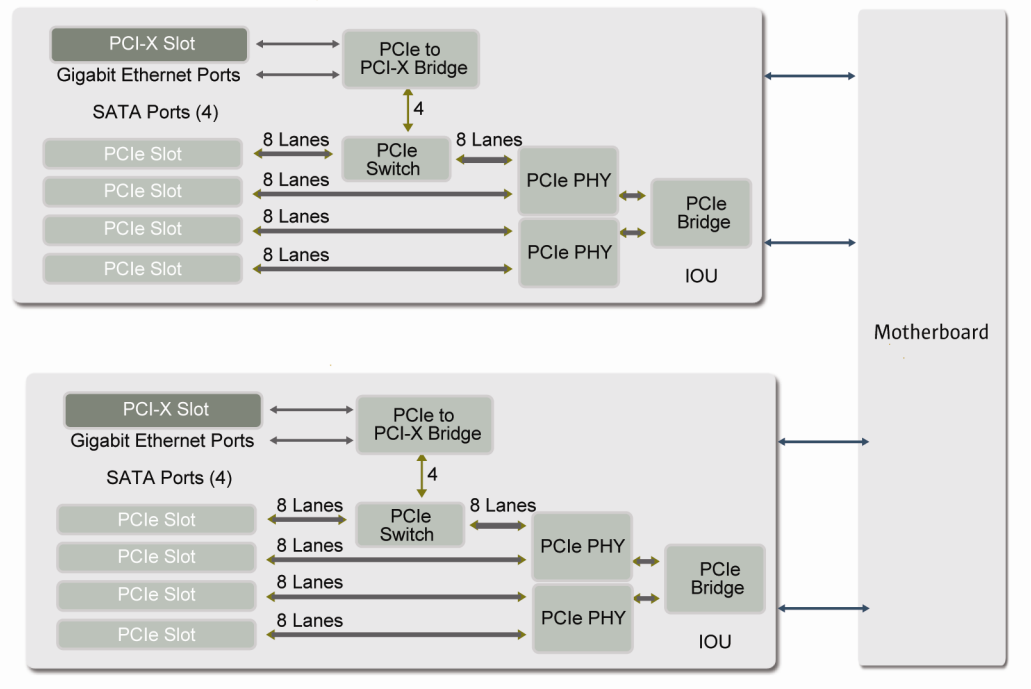

Figure 8. SPARC Enterprise M5000 server I/O subsystem

Within SPARC Enterprise M8000 and M9000 servers, the I/O board supports eight PCI Express slots (Figure 9). Each pair of slots (0,1), (2,3), (4,5), and (6,7) feed into an I/O channel. Each pair of channels feed into a Oberon bridge chip that connects to the XB board.

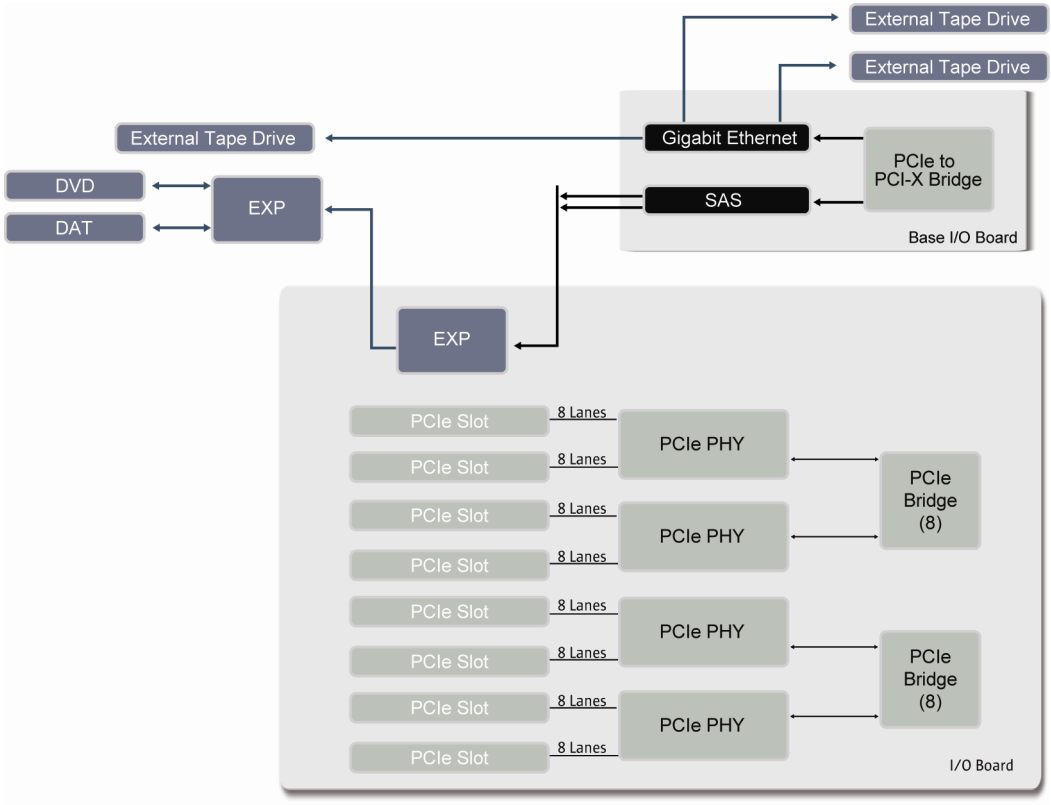

Figure 9. SPARC Enterprise M8000 and M9000 I/O Board

I/O Channel Limitations

The placement of I/O devices can impact available bandwidth. A total bandwidth limit exists on each I/O channel. As a side effect, when two high-speed devices — such as disk arrays or 10 Gb Ethernet cards — are each using one of the slots in a slot pair, the maximum available bandwidth is significantly less than what might be expected from measuring either device alone. No such limitation is observed when devices are placed in different channels. As a general recommendation, when fewer slots are needed, every other slot should be used instead of slots in a series. For example, if four disk arrays are attached to I/O cards in slots 0, 2, 4, and 6 on a SPARC Enterprise M8000 or M9000 server, higher maximum throughput can be obtained than if slots 0, 1, 2, and 3 are used. On Oracle's SPARC Enterprise M4000 and M5000 servers, utilizing slots 2 and 4 results in the best potential throughput.

Migrating to SPARC Enterprise M-Series Servers from Existing SPARC® Platforms

Two very important facts help simplify migration to SPARC Enterprise M-Series servers:

- SPARC64 processors provide full SPARC V7, V8, and V9 binary compatibility.
- The Oracle Solaris operating system that runs on SPARC Enterprise M-Series servers is the same Oracle Solaris operating system that runs on the remainder of Oracle's product line, and binary compatibility is guaranteed.

Applications that run on older systems with SPARC® processors and Oracle Solaris can readily execute on SPARC Enterprise M-Series servers. As described in the next section, Oracle Solaris 10 includes a number of enhancements to help these applications reach maximum performance on SPARC Enterprise M-Series servers.

Oracle Solaris Improvements for SPARC Enterprise M-Series Servers

Three specific projects within Oracle Solaris 10 help optimize the performance of applications on SPARC Enterprise M-Series servers — chip multithreading (CMT) scheduling, non-uniform memory access (NUMA) scheduling, and large-scale optimization.

CMT Scheduling

CMT scheduling deals with issues that arise from running more than one thread of execution on a processor core. When there are fewer active threads to run than there are processors, the best throughput is obtained by spreading the threads evenly over the different cores. With proper thread distribution, no core is left idle at times when any particular core is supporting two threads. To achieve this result, the Oracle Solaris 10 scheduler automatically assigns threads to processors in an even distribution. In addition, Oracle Solaris internals take advantage of the `SLEEP` and `SUSPEND` instruction extensions provided by the SPARC64 VI and VII/VII+ architectures. When a processor strand has no task to run and enters the idle loop, a `SLEEP` instruction executes. Every 1.5 microseconds, the strand wakes up, checks for work, and goes back to sleep if no work exists. Because the strand is not executing instructions during this time, the other strand that shares that core gains 100% of the available processing cycles. In addition, during longer spin or delay loops in the kernel, the `SLEEP` instruction is used to minimize the impact on the thread that shares that core. With the help of the Oracle Solaris these activities occur automatically without any user action required.

NUMA Scheduling

NUMA scheduling helps mitigate issues relating to memory placement and the thread assignment to processors. Within the topology of high-end SPARC Enterprise M-Series servers, processors and memory on the same system board are closer to each other than the processors or memory on other system boards. In Oracle Solaris terms, the processors and memory within the same system board make up a locality group. The Oracle Solaris scheduler assigns threads on initial creation to locality groups, keeping a balanced load across all the locality groups in a domain. The memory allocator

prioritizes the allocation of memory to a thread on a processor from the same locality group. Through these mechanisms, Oracle Solaris can help increase high-speed local memory traffic and reduce slower remote memory traffic. Since the locality group topology is defined at system boot and adjusted automatically during any dynamic reconfiguration, the Oracle Solaris 10 scheduler can continuously maintain adherence to these policies.

Large Parallel Systems

Large parallel systems provide fresh challenges beyond those previously encountered in smaller parallel systems. As systems scale up, smaller and smaller sequential regions limit parallel performance — an argument known as Amdahl's law. Based on this principle, Oracle Solaris benefits from continual improvements in the area of low level systems internals. Recent Oracle Solaris improvements include changes to mutex handling and clock-tick processing.

Mutex optimization

Oracle Solaris utilizes many mutexes to properly synchronize and coordinate memory management, processes, I/O, and other support tasks. In addition, Oracle Solaris provides a user-level interface to a generalized mutex mechanism to allow applications to have the benefit of optimized mutex algorithms. When the frequency of mutex collisions is low — as typical on small systems — the mutex methods introduce only limited effects on overall system performance. As system size grows, the likelihood of another thread holding a mutex when a second or third mutex attempts to access it grows.

To minimize the performance limitations of heavy mutex contention, Oracle Solaris applies a backoff algorithm before retrying contended locks. In Oracle Solaris 10 7/07 (update 4), the backoff algorithm was adjusted and tuned specifically for SPARC Enterprise M-Series servers to provide a substantial increase in mutex throughput under heavy load. Further improvements and innovations to the backoff algorithm for all platforms were added for Oracle Solaris 10 10/08 OS (update 6). These improvements include dynamic adjustments to backoff rate based on system size and mutex usage rate. Due to the many system activities that use mutexes, these improvements help overall system performance even for applications that do not directly use kernel mutexes. The larger the number of strands supported by a domain, the greater the benefits of the improved mutex backoff algorithms.

Scalable Clock-Tick Processing

An important part of Oracle Solaris involves clock-tick processing. In traditional UNIX®-derived systems, a clock interrupt occurs at regular intervals, and is used by the operating system. Each processor has a task queue that includes a scheduler entry when the current task on the processor may need to be rescheduled, as well as "call-out" events where an application or kernel function has requested a delayed "call-out" function to be executed at or after a specific time. The default clock-tick for Oracle Solaris is one one-hundredth of a second. Prior to SPARC Enterprise M-Series servers, clock-tick processing for all processor queues executed on a single processor, typically processor 0. For the largest systems — under some loads — instances were observed where a single processor could not process all the necessary actions for all of the processors' event queue. When that excess load condition was sustained, some events could be delayed indefinitely, resulting in a cascade of negative effects. In Oracle Solaris 10 7/07 OS (update 4), the clock processing code became more efficient to

allow greater numbers of processors to be handled by a single thread. With Oracle Solaris 10 10/08 (update 6), the clock processing code became multithreaded. Under this new model, larger systems can utilize several threads to handle the clock-processing activity. These improvements mean that the clock interrupt load has been spread to run on several processors — changing over time to share the overhead evenly. As a result, clock-tick processing can be handled more efficiently.

Timed Event Processing

Modifications to the processing and scheduling of call-outs within the Oracle Solaris kernel is another means to improve application performance on very large systems. Call-outs are time-based events used when an application needs an action to occur at a specified time in the future. Call-outs are commonly utilized in networking code for detecting dropped packets and in applications such as those written to run on the Java™ platform since it self-schedules threads.

There are generally two types of call-outs — those that expect to be triggered and those that expect to be canceled before the trigger time. Call-outs that expect to be triggered might be thought of as events such as calendar wakeup calls or commands to harvest the contents of an I/O buffer, or other time-based event processing. Those call-outs that expect to be canceled are usually error condition warnings. For example, when TCP sends a packet, a call-out event is posted to occur at a certain time. If the packet is acknowledged before the trigger time, the call-out is canceled. If the packet is not acknowledged within the specified time, then TCP sends a replacement package.

Increasing the number of processors raises the rate at which call-out events need to be scheduled and handled. In large systems, contention increases on mutexes protecting the related data structures and otherwise adds opportunities for delays. In order to realize consistent performance, the operating system must be able handle a large number of events in a timely fashion.

Oracle Solaris Recommendations

To achieve the best possible performance levels — especially on the larger SPARC Enterprise M-Series server models — Oracle recommends upgrading to the latest Oracle Solaris 10 update release. Oracle Solaris 10 7/07 (update 4) is recommended as a minimum release level for Oracle's SPARC Enterprise M8000 and M9000 servers. This updated version of Oracle Solaris 10 includes basic clock-tick optimization and mutex backoff optimization. In fact, Oracle Solaris 10 7/07 (update 4) is required for systems with domains that include 256 processors.

For systems with more than 128 processors in a single domain, Oracle Solaris 10 10/08 (update 6) or patch 137111-03 for Oracle Solaris 10 7/08 OS (update 5) is recommended. This update includes the generalized self-tuning mutex backoff that maximizes throughput for heavily-contended mutexes. Oracle Solaris 10 10/08 (update 6) also includes support for scalable clock-tick processing, using multiple cores for clock-tick processing as needed on larger systems. This update is required to support domains with over 256 strands.

Solutions that make heavy use of system timer events or produce heavy network traffic can benefit from the call-out scalability optimization first available in Oracle Solaris 10 10/09 (update 8). This

update improves the accuracy of timer events under heavy load and is especially recommended for solutions that adjust clock tick resolution.

Advanced Options for Maximizing Performance

While the default settings of Oracle Solaris are designed to support good application performance, a number of advanced tuning steps can help further maximize throughput. Many reference materials exist that describe general methods for application optimization. This section addresses some specific topics that may not be covered by broader discussions.

Utilizing the Latest Compilers

SPARC Enterprise M-Series servers are based on the SPARC64 VI and VII/VII+ processors that include instruction pipelines with different timing from the earlier UltraSPARC® III and IV processors. While completely compatible from a program correctness point of view, there are subtle differences in inter-instruction latencies, L1 latencies, and similar low-level details. The latest Oracle compilers can be targeted to optimize application performance for SPARC Enterprise M-Series servers while still supporting excellent results on other platforms.

The Java™ Platform

Taking advantage of the most up-to-date Java environment with the latest performance innovations can lead to substantial performance gains. After downloading and installing a new Java platform release, applications can immediately benefit from the latest improvements. For the best results, update the users' Java environment to at least the July 2008 version of the Java Platform, Standard Edition 6, Update Performance release. This version can be downloaded from:
http://java.sun.com/javase/technologies/performance.jsp

Oracle Solaris Studio 12 C, C++, and Fortran Compliers and Tools

The latest Oracle Solaris Studio 12 compilers understand and can take advantage of SPARC64 instruction timing. The usual method for telling the compiler to optimize code for execution on SPARC Enterprise M-Series servers is to use the option `-xtarget=sparc64vi`. Executables created with this setting can still run on other servers with SPARC processors. However, the code is tuned for best execution on SPARC Enterprise M-Series servers.

If the switch `-fma=fused` is also used, then fused multiply-add instructions are generated where appropriate. Use of this switch can double the speed of floating-point multiply-add combinations. Unfortunately, executables created with this complier setting can not run on older SPARC platforms that do not implement the fused multiply-add instruction.

Just utilizing the `-xtarget=sparc64vi` option and recompiling with the Oracle Solaris Studio 12 compiler can result in substantial improvement in performance for many applications. The performance gains are due to both machine specific optimizations and cumulative improvements in the Oracle Solaris Studio 12 compiler technology.

The Oracle Solaris Studio 12 compiler patch is available from Oracle for free download by Oracle's Sun Developer Network (SDN) members. More information about obtaining the latest Oracle Solaris Studio 12 downloads and patches can be found at: *http://www.oracle.com/us/products/tools/050872.html.*

Prefetch Tuning

At higher optimization levels, Oracle Solaris Studio 12 compilers automatically take advantage of prefetching. Unfortunately, the complier cannot always provide the optimal amount of prefetching due to application and system variations. If the prefetches are not far enough in advance of the use of a cache line, the data may not arrive in time, and performance can suffer as a result. If too many prefetches are issued, then the read or write queue fills and some prefetches are dropped completely. When an application is ready to use those dropped prefetches, the full memory latency is incurred with no overlap — a situation even more detrimental to performance. Finding the best balance for those two issues can require some tuning with compiler options. The Oracle Solaris Studio 12 compilers make careful, conservative prefetch optimizations to minimize the chance that prefetches cause an application to run slower due to the prefetches using excess memory bandwidth. For some applications, further improvements can be made by directing the compiler to be more aggressive in using prefetching.

For Oracle Solaris Studio 12 compliers, prefetching is only active with optimization level 3 or greater (`-xO3`). For some applications with high bandwidth requirements, performance can be improved by requesting more aggressive prefetching. The compiler option `-xprefetch=latx:<value>` where `<value>` is a simple value such as 2.5 tells the compiler to assume the machine latency is different by that factor from its default assumption. For example, the switch `-xprefetch=latx:5` tells the compiler to prefetch based on the assumption that the machine latency is five times the default. For some applications, best results are obtained using values of 2, 5, or 8. An optimal setting can be determined by compiling and running an application multiple times with different prefetch values. Then the value which provides the best result can be utilized to generate the production executable.

Scheduling Optimization

The Oracle Solaris scheduler works to spread load evenly over the entire system. In general the Oracle Solaris scheduler is very successful at this task, especially on smaller systems. However, for some applications, especially large parallel ones, even small scheduling anomalies can have large performance impacts. For example, consider a memory-intensive, parallel application that needs to use one strand per core to optimize use of cache and memory bandwidth. Assume the application is targeted for execution on a fully-configured SPARC Enterprise M9000-64 server from Oracle with SPARC64 VII/VII+ processors using the full system as a single domain. Each chip provides four cores and each core includes two strands. From the point of view of the Oracle Solaris scheduler, each strand is a processor. The scheduler in this scenario is dealing with 512 virtual processors.

Since the application desires to use one strand per core, 256 threads are required. In the Open Specification for Multiprocessing (OpenMP), setting the environment variable `OMP_NUM_THREADS` to 256 can achieve this requirement. At this point, the scheduler is now responsible to spread the 256 threads over the 512 virtual processors so that each core has one active strand and one idle strand.

Since the application is known to be memory intensive, if two strands of the same core are executing threads from the application, those strands are likely to run substantially slower. Most parallel applications need to synchronize using barriers from time to time. The effective speed of getting through the barrier is limited by the speed of the slowest thread to reach the barrier. As a result, the overall throughput can be sharply lower if two strands are running on the same core for this memory-intensive application.

Over time, interrupts occur and system threads are scheduled, causing load imbalances. If even 1% of the thread scheduling decisions are suboptimal, one thread out of 256 is mis-scheduled, resulting in placement of two threads on the two strands of the same core. With 256 threads to be scheduled, much of the time, some threads are running slow. As a result, the entire application can be slow in passing through barriers. There are methods for preventing the scheduler from falling into these scheduling glitches, such as processor binding and processor sets.

Processor Binding

With processor binding, the application assigns each thread to a particular virtual processor. The `pbind(1)`, `processor_bind(2)`, or `pset_bind(2)` commands are used to execute this control. For example, executing the command while in a interactive shell binds the shell to the specified processor id.

```
pbind -b <processor_id> $$
```

Any threads spawned by the shell also become bound to that processor id. The `pbind` man page provides more examples. Shell scripts and environment variables can be used to generalize the approach. This method has the advantage of rigidly forcing a particular scheduling order that fits the needs of the application. Processor binding does not require root privileges. However, the binding process fails if the specified processor is not in the current domain or zone. Also, processor binding does not exclude other processes from using the specified processors.

Processor Sets

With processor sets, a group of processors are selected to be used for execution of a specific application. The `psrset(1)` command is used to set up the grouping and start a process in the processor set. With this method, all threads spawned within the processor set continue to be scheduled in the processor set. If a processor is removed from the set or system, the remaining processors continue to execute the application. In this way, processor-set scheduling is more flexible than processor binding. However, processor sets are not always as precise processor binding. The `psrset` command requires root privileges since it excludes some threads from some parts of the system.

Processor Disabling

Another method that avoids having two strands running on a single core is to simply turn off one strand on each core. The man page for `psradm(1)` details how to accomplish this effect. Simply

execute `psradm -f` on all the odd (or even) processors to put the machine in single strand per core state. The psradm command requires root privileges since it can remove processors from active use.

In the case of the hypothetical memory intensive application, processor sets, processor binding, or processor disabling can be used to the same effect. A processor set or processor binding of all the odd (or even) processors results in the application having an exact match of one thread to one strand of each core. This method eliminates the possibility of scheduling two threads on a single core, avoiding the bad performance case. In addition, if one strand is disabled for each core, two strands can clearly not be scheduled on that core.

Determining the method that is best for a particular application or domain is likely to be more an administrative decision than a technical one. As such, the descriptions lie outside the bounds of this paper.

Single vs. Dual Processors per Core

The whole topic of choosing between running with one thread per core or two threads per core is new with multistrand architectures. The purpose of having multiple strands per core is to increase total system throughput, even if individual strands run slower. If the system is lightly loaded, the Oracle Solaris scheduler automatically distributes the threads so that each core only has one active processor. Opportunities for adjusting the default scheduler behavior arrive as the system load increases. There are three well-understood cases where it can be advantageous to disable one strand per core and run the system as a single strand per core machine. These cases can be considered as memory bandwidth-limited, cache-limited, and priority-threaded applications.

- **Bandwidth-limited applications**

  For memory bandwidth-limited applications, a single strand per core requests all the bandwidth that a core can provide. These applications are typically characterized by having large amounts of regular data that is suitable to aggressive prefetching and commonly found in the high-performance computing (HPC) space. For this type of application, the second strand merely competes with the first strand for the available bandwidth without providing any additional throughput. In addition, the second strand shares or reduces the available L2 cache space. As a result, the cache per strand is smaller, a fact that is likely to increase memory accesses and lower throughput. Running in single-strand mode can improve overall system throughput. Using the command `psradm -f` on all the odd (or even) strands places the machine in single strand mode. If that type of application is typical for the system, a system administrator can configure the system initialization scripts so that the `psradm` command is executed during the boot process and the system is always set in single strand mode.

- **Cache-limited applications**

  For cache-limited applications, a certain amount of cache space is needed for the application to run efficiently, commonly called the working set. If an adequate amount of cache is available, then most data accesses by the application hit the cache and memory traffic remains manageable. If less than the working set for the application is available, then there are frequent cache misses and more memory traffic.

With two strands per core, the available cache per strand is half that available with one strand per core. If that reduction in cache space causes the application to no longer be able to fit its working set in cache, then running multistrand mode can provide lower throughput than running in single-strand mode. Often, the easiest way to detect this condition is to compare throughput by running an application with both dual strands per core and a single strand per core, with the same input data. This situation is not particularly common, but when it does occur, significant gains can be had by shifting to single-strand mode. As in the memory-bandwidth limited case, `psradm` can be used to put the machine in single-strand mode.

- **Priority-threaded applications**

  For certain applications, a specific set of threads are more important than most other threads. In databases, the database logger thread frequently carries a heavier task load than other threads, and it can provide the absolute limit on how many transactions per second can be achieved in the system. When this sort of condition sets the system throughput limit, the critical thread can be placed in a two-strand processor set or bound to a single strand. Then the other strand can be disabled using the `psradm -f` command or processor sets. In this way, a processor is dedicated to the critical thread to help realize maximum throughput.

Locality Group Scheduling

In some classes of applications, there is a natural grouping of threads that communicate only among themselves. The Oracle Solaris scheduler often has no way of knowing about this communication affinity. Placing these threads in the same locality group can enhance application performance, since communication is typically done by pipes or shared memory. In either case, as the data moves from one thread to another, if it stays in the locality group, the rate of transfer is faster and the latency of transfer is lower. Processor sets or processor binding can be used to administratively control thread grouping.

Network Tuning

Optimization of network interfaces is a large topic. The importance of network tuning depends strongly upon how the network is to be used. For example, heavy NFS traffic requires different network optimizations from a system that processes many Web connections. Larger systems tend to have more network connections and can benefit more from network optimization. The following reference provides detailed guidance on network optimization:
http://www.solarisinternals.com/wiki/index.php/Networks

System Tools for Understanding System Behavior

Oracle Solaris provides a huge variety of measurement tools for understanding system behavior. Sometimes knowing where to start looking can speed the search for solutions. A few tools that are often helpful are discussed below. Only a brief summary of each command is described here. For more information and options, please see the appropriate man pages for the commands.

Obtaining System Configuration Information

Gathering system configuration information is often the first step in understanding the characteristics of a system. Oracle Solaris and SPARC Enterprise M-Series servers offer a number of built-in tools to help simplify the process of obtaining this information.

Oracle Solaris Commands

The `prtdiag` command is commonly used to gain system diagnostic and configuration information. Executing the `prtdiag` command displays a list of components installed on the system along with full device names. In addition, this command can list the most recent AC power failure, identify failed field replaceable units (FRUs), and provide environmental status information.

The `psrinfo` command can provide detailed information about the processors installed on a server. For example, the command `psrinfo -vp`, prints one line for each configured processor, displaying whether the processor is on-line, non-interruptible, spare, off-line, faulted or powered off, and when that status last changed. The information provided by `psrinfo` is very helpful while setting up the scheduling optimization techniques described in the previous section.

eXtended System Control Facility

SPARC Enterprise M-Series servers include an eXtended System Control Facility (XSCF). The XSCF consists of a dedicated processor that is independent of the server and runs the XSCF Control Package (XCP) to provide remote monitoring and management capabilities. This service processor regularly monitors environmental sensors, provides advanced warning of potential error conditions, and executes proactive system maintenance procedures as necessary. XCP facilitates Dynamic Domain configuration, audit administration, hardware control capabilities, hardware status monitoring, reporting, and handling, automatic diagnosis and domain recovery, capacity on demand operations, and XSCF failover services.

Reporting TLB Misses

The `trapstat` command reports trap activity, either system wide, on the processors in a particular processor set, or on the processors on a specified list. When used with the `-T` option, `trapstat` reports TLB miss information. A sample `trapstat` command is as follows:

```
trapstat -T 10 5
```

Redirecting trapstat output to a file is recommended as the data can be lengthy for large systems. The given sample command reports the TLB miss information including page sizes, over 10 second intervals for a total of five iterations. If high TLB misses are occurring, a need to use large pages may be indicated. The TLB structure of SPARC Enterprise M-Series servers is larger and more flexible than any previous system with SPARC processors. As a result, TLB contention occurs less often than on older systems. Still, TLB contention can significantly impact performance.

Please see the paper "Supporting Multiple Page Sizes in the Solaris Operating System" available at *http://www.sun.com/blueprints/0304/817-5917.pdf* for more details on how to benefit from using large

pages. Additional information on this topic is also contained in the paper titled, "Large Memory Pages in Solaris 10". Please contact your local Sun representative for access to this document.

Tracking Lock Contention

The commands `lockstat` and `plockstat` can help track kernel and user level activity. These tools measure lock contention events, including frequency and timing data for these events. The output from these commands displays the data in decreasing frequency order so the most common events appear first. A common practice is to run the following two `lockstat` commands to measure lock events and detect hot locks.

The following command can be used to measure lock events:

```
lockstat -cPwn 100000 sleep <seconds>
```

The following command is useful for detecting hot locks:

```
lockstat -kIWn 100000 sleep <seconds>
```

The specific arguments used in these examples are described in Table 5.

TABLE 5. LOCKSTAT ARGUMENTS UTILIZED BY THE SAMPLE COMMAND

ARGUMENT	DESCRIPTION
`W`	distinguish events by lock not caller
`C`	coalesce lock for lock arrays (for example, pse_mutex [])
`P`	sort data by (count * time)
`n 100000`	sets the maximum number of data records at 100,000
`sleep <seconds>`	specifies how long to run the lockstat measurement
`W`	distinguish events by caller not by lock
`K`	coalesce program counters (PCs) within functions
`L`	watch profiling interrupt events

The `W`, `k`, and `l` switches allow the second `lockstat` command to measure lock events by routine and detect the routines that contain hot locks. By measuring lock contention both ways, hot routines and hot locks are identified. Having both sets of data helps administrators understand possible lock contention and possible resolutions. The `plockstat` command measures user level lock contention. More details and specific arguments for the `plockstat` command can be found on the `plockstat` man page.

Monitoring System I/O Activity

The iostat command measures and reports system I/O activity. Examining the output from iostat is often helpful in identifying system imbalances in disk I/O or observing whether particular file systems are heavily used. A typical `iostat` command is as follows:

```
iostat -cCdMnxz -l 80 10 200 > iostat.out
```

Descriptions of the specific arguments in this sample command are found in Table 6.

TABLE 6. IOSTAT ARGUMENTS UTILIZED BY THE SAMPLE COMMAND

ARGUMENT	DESCRIPTION
`c`	report % time user, sys, I/O, and idle
`C`	report statistics by controller id
`d`	report number of kbytes/second, transfers/second, and average service time
`M`	use Megabytes not Kilobytes for rates
`n`	report names in descriptive format, such as c0t0d0s0
`x`	report extended disk statistics
`l 80`	report 80 I/O devices, adjust the parameter according to system size
`z`	do not report devices with zero activity

Advanced Tools

As described earlier, certain application types have difficulty capitalizing on the capabilities of multithreaded processors. For these applications, adding more strands does not always increase application throughput. As a result, looking at processor utilization rates can be a misleading measure of system load. If all cores have one strand busy and one strand idle, utilization is reported as 50%. For an application that can not drive work to both strands, 50% utilization may be close to the maximum achievable throughput. A new tool called `corestat` can help clarify utilization rates within multithreaded systems. Special versions of `corestat` exist for different types of processors. Please see the following Web site for more details: *http://cooltools.sunsource.net/corestat/index.html*

Monitoring the low-level behavior of the system processors can help developers realize additional insights related to system performance. The `cpustat` and `cputrack` tools provide access to the processor's internal performance counters, helping facilitate detailed workload analysis. Careful selection of counters can help developers determine many application processing characteristics, including the mix of instructions, average cycles per instruction, L1 and L2 cache miss rates, prefetch rates, and other low level details.

The full set of counters can be found in the Fujitsu SPARC64 VI Extensions document at: *http://www.fujitsu.com/downloads/SPARCE/others/sparc64vi-extensions.pdf* . The performance counters are discussed in Appendix Q, "Performance Instrumentation". Reviewing the man pages for `cpustat` and `cputrack` as well as the performance counters in the SPARC64 VI Extensions document can be a good starting point to understanding detailed data collection and analysis.

Other Tuning Resources

The descriptions provided within this article are just a sample of the available tools for measuring system behavior. Others tools include the Oracle Solaris Dynamic Tracing (DTrace) facility that supports extensive system profiling without the need to recompile applications. While a full discussion of system performance analysis is beyond the scope of this paper, these examples provide a starting point for those new to the performance tools provided within Oracle Solaris.

There are a number of other documents that provide useful insights for application tuning for large systems. For example, the document "Developing and Tuning Applications on UltraSPARC T1 Chip Multithreading Systems", available at *http://www.sun.com/blueprints/1205/819-5144.pdf* includes a large section on general purpose parallel application tuning and is highly recommended further reading. Browsing the collection of performance articles at *http://www.sun.com/blueprints/browsesubject.html#performance* can also be instructive.

Summary

The SPARC Enterprise M-Series servers with SPARC64 processors are enterprise-class systems, capable of delivering sustainable levels of high application performance. Within this family of servers, system sizes range from a single processor, quad-core system up to a 64 processor, 256 core, 512 virtual processor system. This breadth of scale within this product family presents both opportunities and challenges. Some of the newer features offered by Oracle's SPARC Enterprise M-Series servers such as dual strands per core and high processor counts can require some application and system specific tuning to achieve the best possible results. Proper attention to application specific behavior can make considerable difference in observed total system throughput.

For More Information

About the Author

Patrick McGehearty is a Staff Engineer in the Performance Technologies Group. During his five years with Sun, he has focused on Oracle Solaris performance and performance characterization of Oracle's SPARC Enterprise M-Series product line with emphasis on how to get the best performance for customer applications. Patrick is currently working on tackling scalability challenges in Oracle Solaris OS in preparation for ever larger systems. Patrick holds a Ph.D. in Computer Science from Carnegie Mellon University. He also holds two patents relating to the interaction of instruction ordering and memory bandwidth optimization.

Related Resources

REFERENCE	URL
SPARC Enterprise M4000, M5000, M8000, and M9000 Server Architecture	http://oracle.com/us/products/servers-storage/servers/sparc-enterprise/index.htm
Java Standard Edition 6 Update Performance release	http://java.sun.com/javase/technologies/performance.jsp
Oracle Solaris Studio C, C++ & Fortran Compilers and Tools	http://www.oracle.com/us/products/tools/050872.html
Developing and Tuning Applications on UltraSPARC T1 Chip Multithreading Systems	http://www.sun.com/blueprints/1205/819-5144.pdf
Supporting Multiple Page Sizes in the Solaris Operating System	http://www.sun.com/blueprints/0304/817-5917.pdf
Network Optimization	http://www.solarisinternals.com/wiki/index.php/Networks
Sun BluePrints Articles related to system performance	http://www.sun.com/blueprints/browsesubject.html#performance
STREAM: Sustainable Memory Bandwidth in High Performance Computers	http://www.cs.virginia.edu/stream/
Solaris Dynamic Tracing Guide	http://www.sun.com/bigadmin/content/dtrace/

Oracle is committed to developing practices and products that help protect the environment

Performance Considerations for Developers Utilizing Oracle's SPARC Enterprise M-Series Servers
April 2011, Version 1.2
Author: Patrick McGehearty

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Hardware and Software, Engineered to Work Together