An Oracle White Paper
August 2011

Oracle Solaris Operating System: Optimized for Sun x86 Systems in the Enterprise

Executive Overview

This document is intended as a technical guide for developers and system administrators that want to understand the precise details of how Oracle® Solaris and Sun x86 systems from Oracle that use the Intel® Xeon® processor E7 family can improve your enterprise application solution environment. It includes brief technical descriptions of specific features and capabilities that can be implemented in Oracle Solaris to optimize the specific functionality of Intel's new processor family in the areas of scalable performance, advanced reliability, security, power efficiency, and cost-effective virtualization. An appendix provides additional technical information on optimizing Oracle Solaris for the Intel Xeon processor E7 family.

Introduction

Oracle and Intel, as part of a broad strategic alliance, have been working together to ensure that Oracle Solaris is optimized to unleash the power and capabilities of current and future Intel Xeon processors, which are used in Sun x86 systems. The Sun Fire server family from Oracle is setting new standards for x86-based systems with leading performance, outstanding scalability, and unmatched reliability, availability, and serviceability (RAS). Sun Fire servers are powered by up to eight Intel Xeon processors and up to two terabytes of memory, and include comprehensive, enterprise-class features. Oracle and Intel have made significant advances to optimize Oracle Solaris for Intel Xeon processor-based systems, and have worked closely to develop new capabilities that are part of the Intel Xeon processor E7 family (codename Westmere-EX). Some examples of this include:

- **Scalable performance.** Oracle Solaris enhances Intel® multicore processor capabilities—up to eight processor sockets, each with up to ten cores and two threads per core—as well as Intel® Hyper-Threading Technology, Intel® QuickPath Technology (Intel® QPI), and Intel® Turbo Boost Technology. Additional optimizations improve memory and network performance.
- **Advanced reliability.** The Oracle Solaris Fault Management Architecture (FMA) integrates with the Intel® Machine Check Architecture (MCA) Recovery features, enabling systems to automatically monitor, report, and recover from hardware errors to maintain data integrity and keep mission-critical applications and services online.
- **Power efficiency and utilization.** Oracle Solaris takes advantage of performance-enhanced dynamic power management capabilities of the Intel Xeon processor E7 family.
- **Cost-effective virtualization.** Enhancing Oracle Solaris to take advantage of the latest Intel® Virtualization Technology (Intel VT) features enables the highest consolidation ratios.

Oracle Solaris and Sun x86 systems, powered by the Intel Xeon processor E7 family, deliver a complete, open, and integrated solution for your datacenter.

Figure 1. The Sun Fire X4800 M2 server is Oracle's most powerful and expandable x86 system—it can be equipped with up to eight Intel Xeon processor E7-8800 series CPUs.

The Oracle Solaris Ecosystem

The Oracle Solaris ecosystem consists of Oracle Solaris as well as the Oracle Solaris Studio software development tools, which form the core of a large developer community and a vast portfolio of applications. Oracle Solaris is a proven, industry-leading operating system with features designed to handle enterprise, business-critical operations. Oracle Solaris provides stability, scalability, high performance, and guaranteed forward binary compatibility, with over 50,000 businesses and institutions running over 11,000 applications on Oracle Solaris today. Intel has embraced Oracle Solaris as a mainstream OS and the enterprise-class, mission-critical UNIX® OS for servers based on Intel Xeon processors.

Oracle Solaris includes many unique and innovative technologies that are uncommon to other operating system vendors, including Oracle Solaris ZFS, Dynamic Tracing (DTrace), Predictive Self Healing, built-in virtualization, and independent security verification. Oracle protects your IT investments by guaranteeing that existing Oracle Solaris 8 and 9 applications will run unmodified on Oracle Solaris 10.

The Oracle Solaris ecosystem, deployed on Sun Fire servers with the Intel Xeon processor E7 family, is an outstanding choice for leading-edge enterprise applications. Organizations can leverage Oracle Solaris as the mission-critical, enterprise-class operating system on Intel Xeon processor series systems.

Intel Xeon Processor E7 Family

The latest Xeon processors extend the success of the existing product line. Oracle Solaris engineers have worked together with Intel to optimize Oracle Solaris for these new platforms.

The Intel Xeon processor E7 family (codename Westmere-EX) is designed for systems with two to eight (or more) sockets. It offers up to ten cores and 20 threads per processor—up to 160 threads in an eight-socket system. Features such as Intel Hyper-Threading and Intel Turbo Mode are now supported by the OS in configurations with up to eight sockets, and more. The Intel Xeon processor E7 family also offers more reliability features, including Machine Check Architecture (MCA) Recovery, which enables the system to detect and recover from errors in memory and cache that were previously "uncorrectable" through ECC or other means. The Intel Xeon processor 7500 series can address large amounts of memory, and is built using a 32nm process. The Intel Xeon processor 7500 series is available in Sun Fire x86 systems, including Sun Fire X4470 M2 and X4800 M2 rackmount servers.

Oracle Integration

As shown in Figure 2, Oracle can now offer customers a complete integrated stack, from the applications layer at the top to disk storage systems at the bottom. Oracle is the number one vendor in the top three software segments (applications, middleware, and database), and Oracle Solaris is today the number-one deployment platform for Oracle Database applications in the market. Oracle offers customers a complete top-to-bottom solution that is open and fully integrated.

This paper will provide technical information on how Oracle Solaris at the operating system layer is tightly coupled with the new Intel Xeon processor family in the server layer. This level of integration enables the application layers (applications, middleware, and database) to fully recover from system errors without compromising system stability, resulting in maximum overall system performance.

An optimized Oracle Solaris for the new Intel Xeon processor family can enhance application throughput, improve platform performance, and maximize overall platform reliability and security by taking advantage of a tightly woven integration of the Oracle software stack.

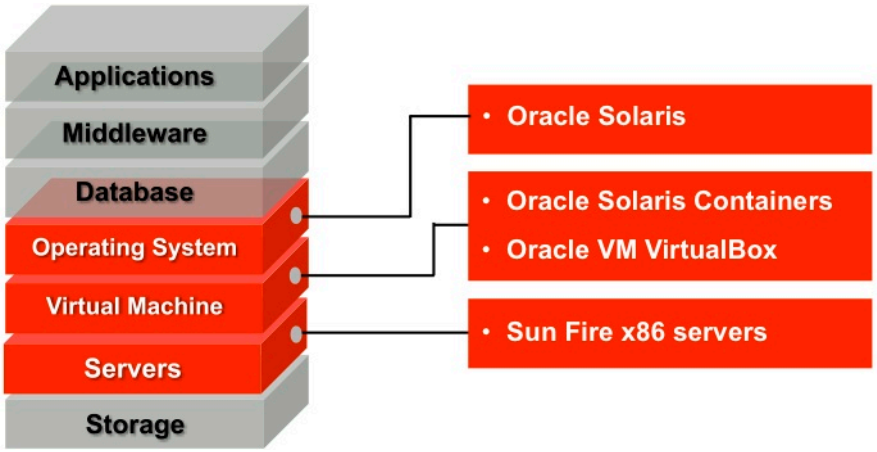

Figure 2. Oracle Solaris and Sun x86 systems are part of a complete, open, and integrated solution.

Intelligent Performance

Successful enterprise-class servers efficiently process CPU-, memory-, and I/O- intensive workloads for middleware and databases. Building on a proven track record, Oracle Solaris takes advantage of the groundbreaking performance capabilities of the Intel Xeon processor E7 family. Significant performance innovation comes from optimizations of the individual cores and the overall multicore microarchitecture, which increase both single-threaded and multithread performance. As a result, the Oracle Solaris kernel and existing single-threaded or multithreaded applications run faster, with no code changes or recompilation necessary.

The Oracle Solaris threading model, and the years of optimization behind it, provides sophisticated performance for commercial applications, outperforming the competition on both application-specific and industry-standard benchmarks. With specific optimizations for the Intel Xeon processor E7 family, Oracle Solaris enables new levels of performance as applications incorporate multithreaded design to increase throughput, responsiveness, efficiency, scalability, and overall performance.

Oracle Solaris has also improved I/O and interrupt scalability when running on the Intel Xeon processor E7 family, enhancing performance and scalability when using PCIe-2.0 interfaces such as 10 gigabit (Gb) Ethernet, 40 Gb InfiniBand, and host bus adapters. Internal testing of common TCP/IP benchmarks show line rate on 10 Gb Intel network interface cards and over 22 Gbps using TCP/IP over InfiniBand quad data rate (QDR) interface cards.

This section also describes other areas where Oracle Solaris and the Intel Xeon processor E7 family work together to provide intelligent performance, including MPO and Intel Turbo Boost Technology.

Memory Placement Optimization (MPO)

Internal testing on systems using Oracle Solaris and the Intel Xeon processor E7 family compared several industry-standard benchmarks with and without Oracle Solaris MPO's Non-Uniform Memory Access (NUMA) optimizations enabled. These results showed that MPO:

- Improves performance across all machines and benchmarks
- Offers the most significant performance gains in memory-intensive workloads, especially on larger machines
- Shows a smaller degree of improvement when measured by a prominent CPU-intensive benchmark

As systems grow larger, with more processor sockets and more memory, the ability of a processor to access memory becomes more challenging—all processors cannot directly access all memory at the same latency. Systems designed with NUMA are designed to enable processors to directly access some memory at the lowest latency, while accessing the rest of the memory with more latency. As shown in Figure 3, the processor in the lower left corner can access the "local" memory next to it in the lower left corner with the least latency. If that same processor needed to access "remote" memory, which is farther away in the upper right corner, there would be more latency.

Multiprocessor systems generally demonstrate some memory locality effects, which means that when a processor requests access to data in memory, that operation will occur with somewhat lower latency if the memory bank is physically close to the requesting processor. Systems using Intel Xeon processor E7 family CPUs store system topology and latency in two tables—Advanced Configuration and Power Interface (ACPI) System Resource Affinity Table (SRAT) and System Locality Information Table (SLIT).

Oracle Solaris takes advantage of the information contained in these tables at boot time and uses the resulting NUMA configuration information to better allocate memory and schedule software threads for maximum performance. Oracle Solaris has a "locality group" (lgroup) abstraction that makes it easier for the operating systems and applications to understand which CPUs and memory are near each other, and to use this information to maximize performance through locality. Oracle Solaris MPO helps optimize latency and bandwidth performance by attempting to ensure that memory is as close as possible to the processors that access it, while still maintaining enough balance within the system to avoid the introduction of bottlenecking hotspots. The lgroup topology models the latency topology of the machine using information from the SLIT and SRAT. This enables Oracle Solaris to optimize for locality with thread scheduling and memory allocation.

Oracle Solaris has long supported MPO, for even the most complex NUMA topologies, to ensure maximum performance in multiprocessor systems. On a system with eight sockets, as illustrated in Figure 3, memory is no more than one or two hops away from any of the eight processors. The extra degree of remoteness can increase the complexity, and MPO is even more important in optimizing performance. Hierarchical Lgroup Support (HLS) improves the MPO feature in Oracle Solaris, enabling the allocation of resources with the lowest possible latency for applications.

Oracle Solaris allocates local resources for a given thread. And, if local resources are not available by default, Oracle Solaris allocates the nearest remote resources. Oracle Solaris takes advantage of the capabilities of the new Intel® QuickPath Interconnect (Intel® QPI) architecture with capabilities such as an optimized scheduler and MPO capability that have proven performance benefits with NUMA architecture systems. In addition, Oracle Solaris can take advantage of Intel Hyper-Threading Technology, executing threads in parallel within each processor core. This helps ensure optimal scheduling of software threads to minimize resource contention and maximize performance.

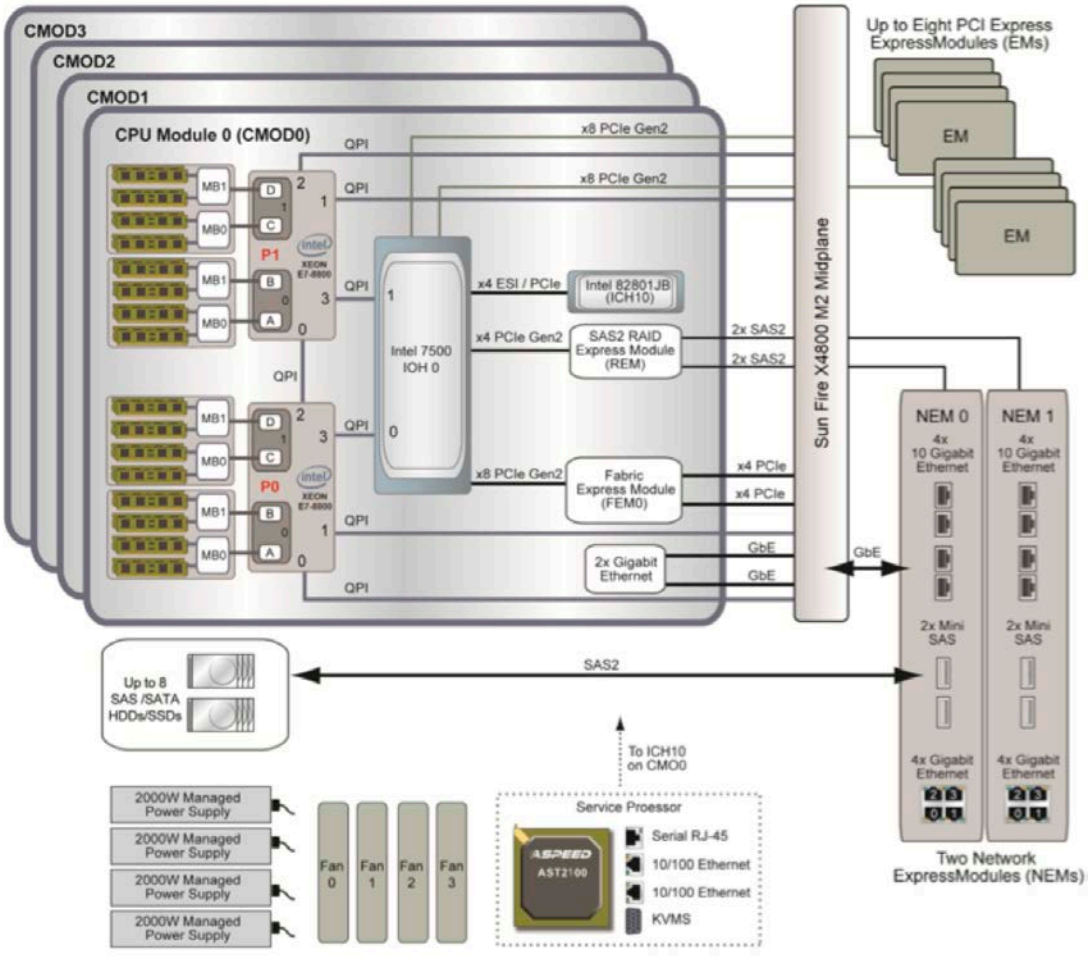

Figure 3. Oracle Solaris understands the physical layout of processing resources, which is especially important when optimizing performance and scalability on eight-socket systems.

Useful Documentation

- *Oracle Solaris Programming Interfaces Guide*: http://download.oracle.com/docs/cd/E19963-01/html/820-0696/eupta.html
- "MPO-aware Oracle": http://wikis.sun.com/download/attachments/128484865/mpo_public.pdf

Intel Turbo Boost

Intel Turbo Boost Technology was first introduced with the Intel Xeon processor 5500 series, and is now available in the Intel Xeon processor E7 family. Intel Turbo Boost Technology converts any available power headroom into higher frequencies. In those situations where Oracle Solaris determines that maximum processing power is required, the Intel Xeon processor E7 family increases the frequency in the active core(s) when conditions such as load, power consumption and temperature permit it. Oracle Solaris and the Intel Xeon processors E7 family work together to determine if cores can be shut down, enabling remaining cores to operate at even faster frequencies.

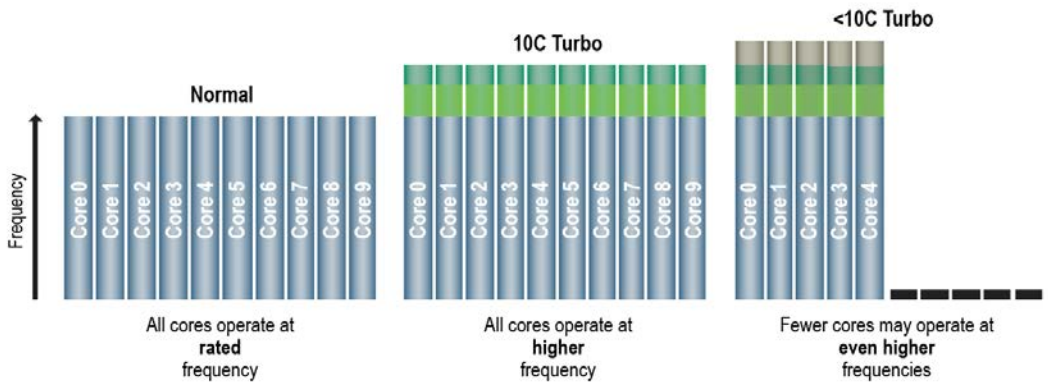

Figure 4. Intel Turbo Boost Technology

By utilizing thermal and power headroom as a performance boost, Oracle Solaris and the Intel Xeon processor E7 family can deliver more work with reduced power consumption. The Oracle Solaris scheduler is "power aware" (see Automated Energy Efficiency, below) and works with the Intel Xeon processor E7 family to maximize performance while minimizing power consumption.

As shown in Figure 4, CPUs typically operate at a fixed maximum frequency regardless of the workload; however, most applications allow the CPU to operate below maximum power. Headroom may also be available if some cores are in idle mode, as pictured. Intel Turbo Boost Technology speeds up the CPU to utilize available power headroom, as needed. The illustration is based on a ten-core Intel Xeon processor E7-8800.

Automated Energy Efficiency

Effective resource management on modern systems requires that there be at least some level of resource manager awareness of the heterogeneity brought on by varying resource power states, and if possible, some exploitation of it. Features such as power capping can help system administrators manage power utilization. Developers can use PowerTOP to optimize energy efficiency for their applications. System administrators can use PowerTOP to analyze and optimize their systems loads, plus identify inefficient programs.

Engineers from Intel and Oracle have worked together for several years to optimize Oracle Solaris for power management and efficiency when operating on Intel Xeon processors. For example, an innovative Power Aware Dispatcher has been integrated into Oracle Solaris, enabling the Intel Xeon processor E7 family to stay longer in idle states (C-states), have better granularity in power management (P-states), and be more responsive to changes in demand. In our preliminary tests, we have seen a substantial reduction in idle power consumption, lower power consumption at maximum processor utilization, and improved performance when switching between power states.

This section provides a general discussion of power management and power efficiency, as well as several technologies and capabilities available on Oracle Solaris to optimize energy efficiency and performance. Developers will be interested in how PowerTOP, the always-running APIC timer, and the Oracle Solaris Power Aware Dispatcher can help control power usage without affecting performance.

Oracle Solaris Power Aware Dispatcher

The Oracle Solaris kernel dispatcher—the part of the kernel that decides where threads should run—is integrated with the power management subsystem of the Intel Xeon processor E7 family. The Oracle Solaris *Power Aware Dispatcher* (PAD) has increased awareness of these Intel Xeon processors, allowing threads to be scheduled according to the power state of the processor. Workloads can be efficiently utilized on available hardware threads, with benefits for shared pipelines, shared caches, and shared sockets. The Oracle Solaris PAD is able to determine which processor resources are being used by the operating system and which are not. The Oracle Solaris kernel now has the ability to utilize those parts of the processor that are active and to continue to avoid doing work on those parts that are powered down. With support for deep idle CPU power management (deep C-states) in the Intel Xeon processor E7 family, Oracle Solaris can dynamically place uninitialized cores into a state where they consume very little power. Any available power headroom can be applied to increase the performance of other cores through Intel Turbo Boost Technology. Together, these two capabilities result in greater power efficiency without losing any performance. This integration enables the Intel Xeon processor E7 family to enter into a deeper C-state, and do so without latency. These capabilities are enabled by default.

In our tests, we have seen a substantial reduction in idle power consumption, lower power consumption at maximum processor utilization, and improved performance when switching between power states.

Useful Documentation

Project Tesla: CPU Power Management: http://download.oracle.com/docs/cd/E19963-01/html/821-1462/powertop-1m.html

PowerTOP

One fruitful technique for understanding system power utilization is observing system-wide resource utilization behavior at what should be zero utilization (system idle). By definition, the system isn't doing anything useful, so any software that is actively consuming CPU cycles is instantly suspect. *PowerTOP* is an open source utility developed by Intel specifically to support this methodology of analysis. Running the tool on what should be an otherwise idle system, ideally one would expect that the system's processors are power-managed 100 percent of the time. In practice, inefficient software (usually doing periodic time-based polling) will keep CPUs fractionally busy. PowerTOP shows the extent of the waste, while also showing which software is responsible. System users can then either report the observed waste as bugs or elect to run more-efficient software.

PowerTOP is a command line tool that shows how effectively a system is taking advantage of the processor's power management features. The application observes the system on an interval basis and displays a summary of how long the processor is spending (on average) at each different state. PowerTOP also reports what kind of system activity is causing the operating system to transition the processor to higher or lower operating levels. This report allows users to understand which applications on their system are affecting power consumption.

Ideally, an un-utilized (idle) system will spend 100 percent of its time running at the lowest frequency and idle states. But because of background user and kernel activity (random software periodically waking to poll status), idle systems typically consume more power than they should. PowerTOP shows what events are causing the system to wake up, and how often.

PowerTOP leverages the Oracle Solaris DTrace framework to quickly and safely analyze the system without impacting performance or service levels. The tool is based on DTrace programs that observe processor idle states transitions, frequency state transitions, and system activity that can lead to changes in power consumption. The information gathered about processor idle and frequency states transitions is displayed at the top of the terminal, while a listing of the most recent and frequent events is displayed below (Figure 5).

By utilizing DTrace, PowerTOP even allows a developer or administrator to observe Intel Turbo Boost Technology operation. Normally, it is difficult to know how fast a system with Intel Turbo Boost Technology is running, as this capability is determined by power systems in the Intel Xeon processor E7 family.

PowerTOP can be downloaded from the package repository using Solaris 11 Express by installing the `pkg:/diagnostic/powertop` package.

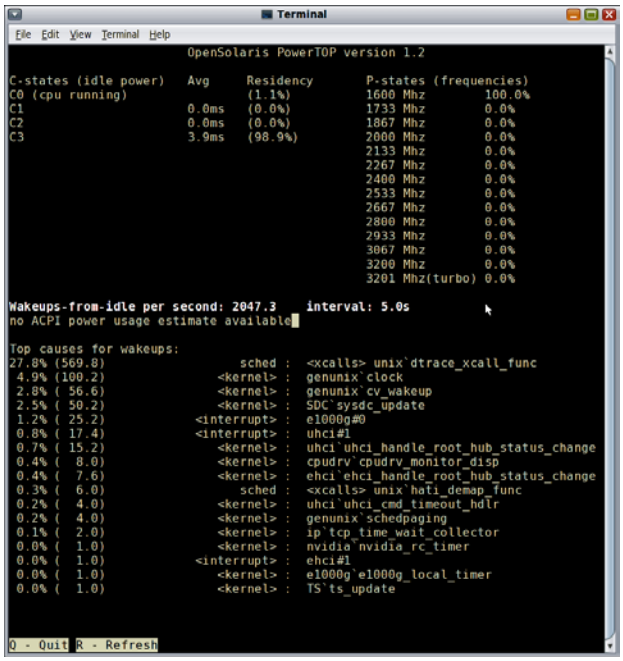

Figure 5. PowerTOP for Solaris

Power Budgets and Power Capping

Systems built using the Intel Xeon processor E7 family are capable of using a power budget feature to help you better plan and manage the power required for your datacenter. *Power capping* is achieved by limiting the maximum frequency at which the CPUs run. Power budget features, such as power capping, can help datacenter administrators control the peak power used by their systems.

The ACPI specification describes, among other things, the notion of CPU Performance States (P-states), which are defined combinations of voltage and core frequency that have been specified by the CPU manufacturer. Lower operating core frequencies allow a reduction in core voltage, with the result being an often dramatic reduction in power. The reduction in power moving from one P-state to the next lower one is proportional to the frequency change and proportional to the square of the voltage change.

The mechanism through which power capping affects system power consumption is ACPI-defined CPU performance states. Power capping works by modifying CPU P-states to allow the cores to run as fast as possible while maintaining a total system power consumption that is at or below a level prescribed by the power policy currently in effect.

The power capping capabilities can implement both hard limit (no spikes allowed over the power limit) and soft limit (spikes are allowed within a time limit) consumption requests from the power capping metrics. Initial testing shows that there can be up to a 25-percent power reduction between the maximum P-state and the minimum P-state on a 130W processor.

The user controls power capping by setting the target power consumption. The power management and capping features are a capability within the Integrated Lights Out Manager (ILOM) service processor that monitors measurable values, determines whether adjustments are needed in consumed power, and then applies power capping methods to achieve the adjustment. A closed-loop control architecture, in which ILOM software monitors system power consumption and modifies CPU P-states, allows the CPU to run as fast as possible while maintaining a total system power consumption that is at or below a level prescribed by the power policy currently in effect.

Useful Documentation

Sun Integrated Lights Out Manager (ILOM) 3.0 Feature Updates and Release Notes: http://download.oracle.com/docs/cd/E19860-01/E21444-01/E21444-01.pdf

Always Running APIC Timer

An OS typically uses several timer devices for different purposes. Oracle Solaris relies on advanced programmable interrupt controller (APIC) timers onboard the CPU to schedule interrupts at some future point in time—the cyclic subsystem. ACPI allows these timers to stop when a core goes into deep C-states.

The Intel Xeon processor E7 family provides an always-running APIC timer (sometimes known as ARAT) that no longer requires the set up and transfer of timer data to the HPET in any C-state conditions. The ARAT is constant in deep C-states, while the previous APIC timer is not. A significant benefit of this capability is that the Oracle Solaris scheduler may choose a deeper C-state since the transition latency in and out of deep C-state is significantly reduced.

Useful Documentation

Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1 http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html

Reliability

Protecting data integrity is a significant factor in system reliability. In the Intel Xeon processor E7 family, there are numerous processor, I/O, QPI, and memory reliability features. Protecting data integrity involves the prevention of data errors first, then the detection of data errors that might occur, and finally the containment of any corrupt data to keep if from poisoning other data in the system.

Oracle Solaris provides a proven architecture for building and deploying systems and services capable of *Predictive Self Healing*, which automatically diagnoses, isolates, and helps resolve hardware and application faults. Oracle Solaris Fault Manager is a key component of Predictive Self Healing. System administrators will greatly benefit from these reliability features.

Oracle Solaris Fault Manager receives data relating to hardware and software errors and automatically diagnoses the underlying problem. Once diagnosed, Oracle Solaris Fault Manager automatically responds by offlining faulty components. Oracle and Intel worked together to extend these capabilities to systems based on the Intel Xeon processor E7 family, including chipsets and memory subsystems.

The Intel Xeon processor E7 family adds new, mainframe-inspired reliability, availability, and serviceability features such as Machine Check Architecture (MCA) Recovery. Intel MCA Recovery enables the system to detect and recover from errors in memory and cache that were previously "uncorrectable" through error-correcting code (ECC) or other means. MCA accomplishes this by first detecting and containing errors before the data is consumed by an application, and then by working with Oracle Solaris to determine the best course of action to keep the system and application running. This ability enables those components to be replaced during planned maintenance cycles and allows IT to extend the time between these cycles. This advanced recovery capability means that systems based on the Intel Xeon processor E7 family are able to recover and keep running in situations where other x86-based systems would not.

A critical element to enterprise-class availability is reliable data subsystems. Oracle Solaris ZFS provides unparalleled data integrity, capacity, performance, and manageability. ZFS provides high-resiliency features, such as metadata logging, to guarantee data integrity and speed recovery in the event of system failure. ZFS dramatically simplifies file system administration to help increase protection against administrative error. Oracle Solaris Cluster, built on top of Oracle Solaris, tolerates any failure within a cluster of physical servers by exploiting hardware redundancy and software algorithms. The failure detection and recovery extend the reliability of the Intel Xeon processor E7 family configurations to support end-to-end business continuity.

Oracle Solaris FMA for Intel Xeon Processors

On Oracle's Sun x86 systems, Oracle Solaris works with the Service Processor (SP) to detect and correct faults. Most notably, Oracle Solaris assists in the area of recovery and isolation of problematic resources. For example, the SP detects and diagnoses all processor uncorrectable errors (UEs). Correctable errors (CEs) are remanded to Oracle Solaris, where they can be offlined by Oracle Solaris FMA if CEs occur too frequently. Also, Oracle Solaris FMA captures the error state on CPU UEs and reports it upon the next Oracle Solaris restart.

Oracle Solaris has the capacity to offline individual processor strands (no further software threads are scheduled on the affected strand), retire individual pages of memory (8 KB granularity), and cease using problematic I/O devices (configurations and active usage permitting).

Oracle Solaris FMA works with the Intel Xeon processor E7 family to support several capabilities, including:

- Memory Topology: For DIMM diagnosis and page retire, FMA requires a memory topology. For the Intel Xeon processors E7 family, the memory topology is read directly from the memory controllers on the system via the `intel_nhm` driver, and it is post-processed by FMA's topology enumerators.
- Diagnosis Rule updates: This capability provides coverage for e-reports, notably when the QuickPath detects errors.

Useful Documentation

"Oracle Solaris 10 Update 9: FMA Fixes" blog:
http://blogs.oracle.com/sdaven/entry/s10u9_fma_fixes

Security

New AES instructions significantly reduce CPU overhead when using IPsec. Internal testing shows that when IPsec is enabled, there is better than a 50-percent decrease in CPU utilization in a system based on the Intel Xeon processor E7 family compared to a similar system based on the previous generation of Intel Xeon processors. The comparison used a well-established benchmark that measures network performance and CPU utilization. IPsec was implemented using the Solaris Cryptographic Framework, as described in this section.

The Oracle Solaris Cryptographic Framework implements several industry-standard security technologies, including the PKCS #11, Java Cryptographic Extension (JCE), OpenSSL, and Network Security Services (NSS). It provides cryptographic services to users and applications through commands, a user-level programming interface, a kernel programming interface, and user-level and kernel-level frameworks. The Oracle Solaris Cryptographic Framework can provide performance and security benefits to both system administrators and developers.

The Intel Xeon processor E7 family (codename Westmere-EX) incorporates hardware instructions that directly implement the Advanced Encryption Standard (AES) crypto algorithm in hardware. Intel® Advanced Encryption Standard Instructions (AES-NI) provide significant performance improvements over the previous implementation, which was hand-coded machine language. For users and services that utilize the Oracle Solaris Cryptographic Framework, performance is improved when these AES instructions are used.

The Oracle Solaris Cryptographic Framework provides cryptographic services to applications and kernel modules in a manner seamless to the end user, and brings direct cryptographic services, like encryption and decryption for files, to the end user. The user-level framework is responsible for providing cryptographic services to consumer applications and the end-user commands. The kernel-level framework provides cryptographic services to kernel modules and device drivers. Both frameworks give developers and users access to software-optimized cryptographic algorithms. Additional detail is available in the Appendix B.

For applications that utilize any of the above-mentioned Cryptographic APIs, performance of AES instructions are automatically improved without recompilation. For applications that use a private cryptographic library, recompilation, or linking to one of these APIs will ensure that full hardware acceleration of AES cryptographic routines is achieved. Note that many system services in Oracle Solaris, such as IPSec/IKE and Kerberos authentication, already take advantage of the Cryptographic Framework and automatically use the hardware acceleration provided by the Xeon processors.

Beyond improving performance, the new instructions help address recently discovered side channel attacks on AES. AES-NI instructions perform the decryption and encryption completely in hardware without the need for software lookup tables. Therefore using AES-NI can lower the risk of side-channel attacks as well as greatly improve AES performance.

Useful Documentation

- Intel Advanced Encryption Standard (AES) Instructions Set: software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/
- Chapter 8, "Developing Applications Using the Data Encryption API," of the *Oracle Database Security Guide:* http://download.oracle.com/docs/cd/B28359_01/network.111/b28531/data_encryption.htm
- "How To Encrypt Data in Oracle Using PHP"*:* http://www.oracle.com/technetwork/articles/ullman-encrypt-098895.html

Oracle Virtualization

As the power of today's servers continues to increase way beyond the needs of a single application stack, virtualization has become the must-have technology. Virtualization can harness the full power of modern systems, extracting maximum value from the asset, while delivering an IT landscape that can ebb and flow to meet dynamic business needs.

Oracle provides strong capabilities for desktop, server, storage, and network virtualization. Enterprise users need choice when it comes to server virtualization and consolidation, and flexibility with respect to application, OS, and network virtualization methods. Systems that are designed using the Intel Xeon processor E7 family can provide more processing resources that are ideal for virtualized computing environments. Oracle's virtualization strategy offers the most comprehensive desktop-to-datacenter product portfolio, with integrated management and support of the full hardware and software stack, from applications to disk.

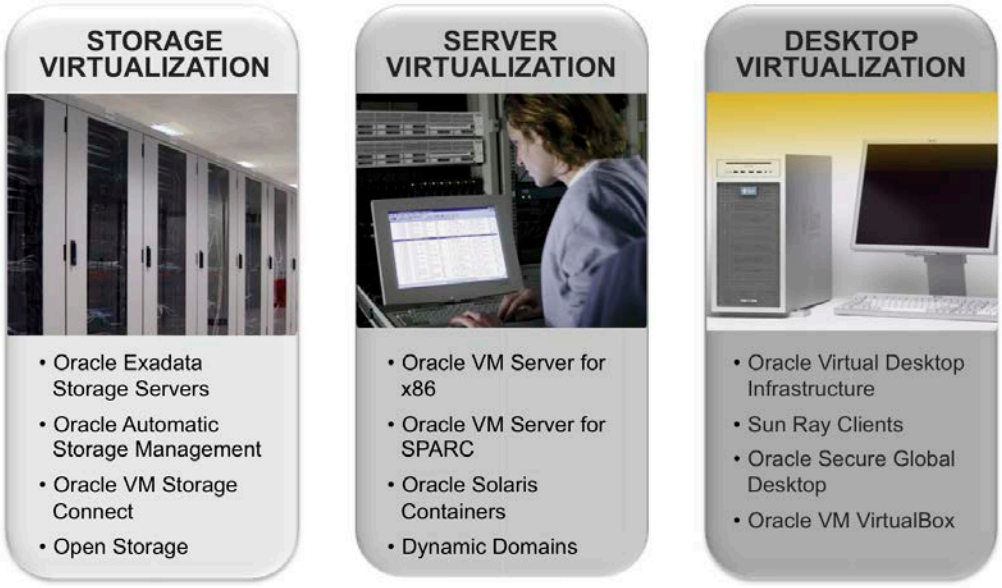

Figure 6. Oracle offers the industry's most complete virtualization portfolio.

Oracle VM Server virtualization and management consists of Oracle VM Server, open source server software, and an integrated Web browser-based management console, Oracle VM Manager. Oracle VM provides an easy-to-use, feature-rich graphical interface for creating and managing virtual server pools running on x86-based systems across the enterprise. Oracle VM supports paravirtualized guest operating systems. Oracle VM also supports hardware-virtualized kernels, and is optimized for the Intel VT architecture.

Intel Virtualization Technology (Intel VT)

The Intel Xeon processors E7 family includes Intel Virtualization Technology (Intel VT), which delivers even more efficient virtual machine operation with optimizations that enable a higher level of performance and reliability. This includes support for CPU (VT-x), IOH (VT-c), and direct I/O (VT-d). These new features enable improved software performance, security, and reliability in virtualized environments. The Intel Xeon processor E7 family delivers faster virtualization performance by reducing transition latency into and out of virtual machine mode, as well as reducing the number of required transitions into and out of the virtual machine mode. In addition, the Intel Xeon processor E7 family offers improved I/O virtualization as part of the core logic chipset, and enhanced I/O performance through direct assignment of a device to a virtual machine. Fast, reliable, comprehensive, hardware-assisted virtualization capabilities contribute to better power efficiency, improved reliability, and increased asset utilization.

Oracle Solaris Virtualization Capabilities

Oracle Solaris Containers provide virtualization capabilities within the operating system layer, enabling massive scalability of resources within Oracle Solaris. A container is a virtualized operating system environment created within a single instance of Oracle Solaris. Applications within Oracle Solaris Containers are isolated, preventing processes in one Oracle Solaris Container from monitoring or affecting processes that are running in another Oracle Solaris Container.

Oracle Solaris Containers enable Intel Xeon processor E7 family systems to have native, bare metal performance with very little overhead and much more efficient overall system utilization, while supporting multiple virtualized environments.

Oracle Solaris Containers consist of two distinct, yet complementary, parts—Oracle Solaris Zones and Oracle Solaris Resource Manager. Oracle Solaris Zones create independent environments where applications can run. Oracle Solaris Resource Manager allocates resources to each of these environments. A zone with or without resource management is an Oracle Solaris Container, and vice versa. However, an optimized Oracle Solaris Container combines both Oracle Solaris Zones and Oracle Solaris Resource Manager.

Oracle Solaris Containers are great for running multiple instances of Oracle Solaris-based workloads. However, today's datacenters are inevitably heterogeneous. So how can an Oracle Solaris-based Intel Xeon server accommodate Windows and Linux workloads? The answer is Oracle VM VirtualBox.

Oracle VM VirtualBox

Oracle VM VirtualBox is a high-performance hypervisor for x86 platforms. Designed to augment the capabilities of the underlying platform, Oracle VM VirtualBox extends Oracle Solaris so that it can run powerful Windows and Linux virtual machines alongside Oracle Solaris workloads on the same server.

Oracle VM VirtualBox supports a broad range of guests, from older Windows Server 2003 or Red Hat 4 through to the latest Windows Server 2008 and bleeding edge Linux distributions. Both 32- and 64-bit versions of operating systems are supported and Oracle VM VirtualBox requires no specialized drivers to be installed.

Performance is paramount in a virtualization platform. Oracle VM VirtualBox extracts maximum performance from the hardware by using the latest Intel VT capabilities of the Intel Xeon processor E7 family.

One of the founding principles of the Oracle VM VirtualBox hypervisor design is to run the guest code directly on the metal. With older processors, this approach results in traps, which have to be dealt with by the Oracle VM VirtualBox Virtual Machine Monitor (VMM), thereby slowing down overall guest execution speed. But with VT-x hardware assistance, there is much less need for VMM intervention and the associated complex, compute-intensive software translations.

A second example of performance gain using VT technology is in the area of rapid context switching between guests and hosts. Each guest operating system is largely unaware that it is running in a virtual machine and, therefore, manages the system hardware resources as though it has exclusive access to them. For example, a Windows Server 2008 guest will manage Page Descriptor Tables, CPU registers and Interrupt Tables as though it were the sole owner of these resources. When a context switch back to Oracle Solaris occurs, the exact state of the Windows guest must be remembered in order that its state can be restored when execution resumes. Historically, some of these operations, such as remembering Page Table state, have taken some time. But with the introduction of Extended Page Tables support in VT-x, switching between Page Table states has become trivial and has led to substantial performance gains.

Furthermore, yet another VT feature, Real Mode Support, available in the Intel Xeon processor E7 family, has led to a significant improvement in boot-time performance, speeding up the execution before the guest has switched to protected mode.

Because the underlying host platform is Oracle Solaris, Oracle VM VirtualBox is also able to marshal the unique benefits of the Oracle Solaris platform too. Here are just a couple examples:

- Virtual machines, which are often multigigabyte files, can easily and rapidly be cloned using the ZFS "copy-on-write" attribute.
- Oracle VM VirtualBox can also be used in conjunction with Oracle Solaris Containers, producing a unique combination where the security and isolation of a zone is married to the cross-platform nature of an Oracle VM VirtualBox virtual machine.

By taking advantage of the Intel VT capabilities in the Intel Xeon processor E7 family, Oracle Solaris and Oracle VM VirtualBox provide a very powerful and performant platform for both desktop and server virtualization workloads.

For more information on running Oracle Solaris as a guest on Oracle VM, see Appendix C.

Useful Documentation

- Oracle's Virtualization Blog: blogs.oracle.com/virtualization/
- Oracle VM VirtualBox: www.virtualbox.org/

Developer Tools Optimizations

The enterprise capabilities of Oracle Solaris enable users to utilize the new features of the Intel Xeon processor E7 family. The Oracle Solaris ecosystem, including the Oracle Solaris and Oracle Solaris Studio development tools, offer a compelling value for developers and users who embrace the breakthrough capabilities of latest Intel Xeon processor E7 family. The result delivers a robust choice for both established enterprise datacenters and leading-edge applications.

Oracle Solaris Studio

Oracle Solaris Studio improves both the development process and ultimate performance in multicore application development. It serves as a comprehensive development, deployment, and testing facility for the Intel Xeon processor E7 family offering award-winning compilers (C, C++, Fortran) optimized for the latest multicore architectures, thread analysis tools, compiler auto-parallelization, OpenMP and MPI support, performance analysis tools, multithreaded debugging, and more. Oracle Solaris Studio also comes with an integrated development environment (IDE) tailored for use with the included compilers, the debugger, and the analysis tools. This integrated development environment increases developer productivity with a code-aware editor, workflow, and project functionality. In addition, the parallelizing C, C++, and Fortran compilers; enhanced math routines; and performance analysis tools enable users to maximize the performance of their applications on the Intel Xeon processor E7 family, generating higher ROI from deployment hardware systems.

Oracle Solaris Studio is optimized to extract high performance from the microprocessor architecture of the latest Intel Xeon processor E7 family. These optimizations include:

- Processor-specific instruction selection and instruction scheduling that enables the compiler to select and place the most efficient instruction that the Intel Xeon processor provides.
- Inlining and optimization of common of `libc`/`libm` functions, which reduces the cost of calling these commonly used functions.
- Compact pointer optimization that enables the benefits of 64-bit binaries without the overhead of 64-bit pointers.
- Improved complex arithmetic operations that get improved performance from using SSE3, Supplemental SSE3, SSE4.1, and SSE4.2 instructions.
- Performance tuning of `sunperflib` (Sun Performance Library) and math routines.
- Improved ability of the compiler to perform microvectorization where it identifies regions of code that could be improved using SSE instructions.
- Improved ability to recognize and automatically parallelize loops.
- Ability to generate code targeted for the latest Intel Xeon processor E7 family using processor-specific code generation options for `-xtarget=nehalem` and `-xchip=nehalem`. These options provide appropriate settings for `-xarch`, `-xchip` and `-xcache` values.
- Parallelization of applications using the full OpenMP 3.0 support provided by both the compiler and tools.
- Support for Intel Xeon processor E7 family performance monitoring events in the Performance Analyzer to help tune and optimize application performance.

Detailed information on developer tool optimizations for Intel Xeon processors is included in Appendix E at the end of this document. This includes performance counters, new processor instructions, improved microvectorization, automatic parallelization, instruction selection, and OpenMP development.

Conclusion

Oracle Solaris has continued to demonstrate great success as a mission-critical, enterprise-class OS for scalable performance, advanced reliability, and power efficiency in the datacenter. The introduction of the Intel Xeon processor E7 family in Sun x86 systems is creating an even greater demand for Oracle Solaris in the x86 volume server segment. Oracle Solaris is leveraging more than 20 years SMP expertise for proven performance in a very large multicore processing environment. Sun x86 systems equipped with the Intel Xeon processor E7 family are well-positioned to take advantage of Oracle Solaris, with features specifically engineered for multicore systems.

Developers and system administrators alike can use Oracle Solaris running on systems designed with the Intel Xeon processor E7 family for improved performance, reliability, and throughput. Deployed together, there are many built-in optimizations, and many more places to fine-tune applications and deployments for specific environments and users.

Sun x86 systems from Oracle and Oracle Solaris are both widely recognized as the technologies of choice for enterprise and mission-critical applications. Whether serving large databases, using high-performance computing applications, or consolidating multiple lower-powered servers, systems must scale smoothly and intelligently.

The collaboration of Oracle and Intel, including the joint engineering work from both companies, has already resulted in state-of-the-art development and deployment platforms for environments based on Intel Xeon processors—an ideal platform for mission- and business-critical deployment.

Resources

See the following Oracle and Intel resources.

Get Product Information

- Oracle Solaris: oracle.com/solaris
- Intel Xeon processor E7 family: intel.com/content/www/us/en/processors/xeon/xeon-processor-e7-family.html/
- Oracle's Sun x86 servers: oracle.com/us/products/servers-storage/servers/x86/
- Oracle's Sun Fire X4470 M2 servers: oracle.com/goto/x4470m2/
- Oracle's Sun Fire X4800 M2 servers: oracle.com/goto/x4800m2/

Deep Dive on the Technical

- Oracle Technical Network: oracle.com/otn
- Intel Developer Center: developer.intel.com/design/index.htm

Visit the Community

- Intel Server Room: intel.com/server

- Intel IT Center: intel.com/content/www/us/en/blogs-communities-social.html

Additional Resources

- "Oracle's Sun Fire X4470 M2 Server Architecture: Virtualization Platform for Mission- Critical Applications": http://www.oracle.com/technetwork/articles/systems-hardware-architecture/o11-053-sf-x4470m2-arch-411693.pdf
- "Oracle's Sun Fire X4800 M2 Server Architecture": http://www.oracle.com/technetwork/articles/systems-hardware-architecture/o11-058-x4800m2-arch-423198.pdf?ssSourceSiteId=ocomen
- IDC white paper: "Oracle x86 RAC Servers: Optimized for Rapid Deployments and Operational Efficiency": http://www.oracle.com/us/products/servers-storage/servers/x86/idc-x86-rack-servers-306298.pdf
- Hardware Compatibility Lists for Oracle Solaris: http://www.oracle.com/webfolder/technetwork/hcl/index.html

Contributing Authors

The following people contributed to this white paper.

Oracle

- Performance and scalability: Jonathan Chew, Georg Edelmann, Damien Farnham, Yan Fisher, Colm Harrington, Allan Packer, Mara Roccaforte, Uday Shetty
- Intelligent Power: Bill Holler, Eric Saxe, Rafael Vanoni
- Reliability: Gavin Maltby
- Security: Dan Anderson, Terri Wischmann,
- Virtualization: Andy Hall, Duncan Hardie, Honglin Su
- Oracle Solaris Studio: Nawal Copty, Ikroop Dhillon, Yi Gao, Kuriakose Kuruvilla, Bhawna Mittal, David Seberger
- Oracle Solaris: Chris Baker, Art Beckman, Darrin Johnson, Amour Kwok, Scott Michael, Mike Mulkey, Karen Perkins, Rick Ramsey, Stephanie Scheffler, Cindy Swearingen, Jeff Victor, Larry Wake, Steffen Weiberle

Intel Corporation

- Robert Kasten

Appendices

The following appendices provide more in-depth technical detail on optimizing Oracle Solaris for the Intel Xeon processor E7 family.

A. Reliability—Support for Microcode Updates on Intel Processors

The Intel family of processors has the capability to correct processor errata by loading an Intel-supplied data block, called a microcode, into the processor. The BIOS contains a microcode loader, which is typically responsible for loading the microcode update during the system initialization process.

The ability to work around processor errata in the operating system by applying microcode updates is now available in Oracle Solaris. This support alleviates the need to upgrade a system's BIOS every time a new microcode update is required.

OS microcode updates can be performed in the following ways:

- During the early stages of booting, as each processor is brought online
- Dynamically, on a system that is booted and running

For microcode updates that are performed during the boot process, the kernel locates the corresponding microcode file, if it exists, and then performs the update, if necessary. On a running system, microcode updates are performed by using the command line through `ioctl` to a driver. The ability to perform a live update during run time provides a means for loading critical microcode updates that ensure system integrity, without requiring a reboot or BIOS upgrade.

For runtime updates, the `ucodeadm(1M)` command can be used to perform the following tasks:

- Report processor microcode revision on the processors
- Update microcode on a live system
- Install microcode on a target system to be used during the boot process

To display the running microcode version, use the `-v` option. To update microcode on all cross-call interrupt-ready processors on a live system, use the `-u <microcode-text-file>` option. To install microcode on a target system to be used for the next boot cycle, use the `-i <microcode-text-file>` option. You can specify an alternate path for installing the microcode files by using the `-R <path>` option. When supplying a microcode text file, note that the file name must include the vendor name prefix.

For example:

```
# ucodeadm -i intel-ucode.txt
```

Microcode files can be obtained from the processor vendor. By default, files are installed in `/platform/i86pc/ucode/$VENDORSTR/` directory, where `VENDORSTR` is `GenuineIntel`.

A microcode file has a 48-byte header, followed by the binary code of the size indicated in the header (`uh_body_size`), and optionally a (20+ n* 12)-byte extended signature block, if the microcode file can be used for more than one type of processor.

To automate the steps for updating the microcode during installation and reboot, the `bootadm(1M)` command has also been modified to invoke the `ucodeadm -i` command, if a microcode text file with a more recent timestamp is available in the `/platform/i86pc/ucode/` directory. The timestamp file and the microcode text file on each microcode installation are set to have the same creation time.

In addition to the `ucodeadm` command, runtime microcode updates are facilitated by a driver, `ucode_drv`. The following capability is provided through these `ioctl` commands:

- `UCODE_GET_VERSION`: Obtains the running microcode version
- `UCODE_UPDATE`: Applies a new microcode

The `ucode_drvdriver` is installed in the `/devices/pseudo/ucode` directory, with a symbolic link from the `/dev/ucode` directory.

Useful Documentation

- Man pages for `ucodeadm(1M)`, `psrinfo(1M)`, `psradm(1M)`, and `bootadm(1M)`
- *Intel 64 and IA-32 Architecture Software Developer's Manual,* Section 9-11, "Microcode Update Facilities": www.intel.com/Assets/PDF/manual/253668.pdf

B. Security—Intel AES-NI Instruction Details

The AES is the United States Government's Federal Information Processing Standard for symmetric encryption, defined by FIPS Publication #197 (FIPS197). AES is a block cipher that encrypts a 128-bit block (plaintext) to a 128-bit block (ciphertext), or decrypts a 128-bit block (ciphertext) to a 128-bit block (plaintext).

AES uses a key (cipher key) whose length can be 128, 192, or 256 bits. AES-128, AES-192, and AES-256 process the data block in, respectively, 10, 12, or 14 iterations of pre-defined sequences of transformations, which are also called *AES rounds* (*rounds* for short). The *rounds* are identical except for the last one, which slightly differs from the others (by skipping one of the transformations).

The rounds operate on two 128-bit inputs: *State* and *Round key*. Each round from 1 to 10/12/14 uses a different round key. The 10/12/14 round keys are derived from the cipher key by the *Key Expansion* algorithm. This algorithm is independent of the processed data, and can, therefore, be carried out independently of the encryption/decryption phase (typically, the key is expanded once and is thereafter used for many data blocks using some cipher mode of operation).

The data block is processed serially as follows. Initially, the input data block is XOR-ed with the first 128 bits of the cipher key to generate the *State* (an intermediate cipher result). Subsequently, the State passes, serially, 10/12/14 rounds, each round consisting of a sequence of transformations operating on the State and using a different round key. The result of the last round is the encrypted (decrypted) block.

The new AES-NI instruction set is comprised of 6 new instructions that perform several compute intensive parts of the AES algorithm. These instructions can execute using significantly less clock cycles than a software implementation. Four of the new instructions accelerate the encryption/decryption of a round, and two new instructions are for round key generation. The following is a description of the new instructions.

- AESENC. This instruction performs a single round of encryption. The instruction combines the four steps of the AES algorithm, `ShiftRows`, `SubBytes`, `MixColumns`, and `AddRoundKey`, into a single instruction.
- AESENCLAST. Instruction for the last round of encryption. Combines the `ShiftRows`, `SubBytes`, and `AddRoundKey` steps into one instruction.
- AESDEC. Instruction for a single round of decryption. This combines the four steps of AEA, `InvShiftRows`, `InvSubBytes`, `InvMixColumns`, and `AddRoundKey` into a single instruction.
- AESDECLAST. Performs last round of decryption. It combines `InvShiftRows`, `InvSubBytes`, and `AddRoundKey` into one instruction.
- AESKEYGENASSIST is used for generating the round keys used for encryption.
- AESIMC is used for converting the encryption round keys to a form usable for decryption using the Equivalent Inverse Cipher.

C. Virtualization—Oracle Solaris as a Guest OS on Oracle VM

Oracle Solaris 10 may be installed as a virtual machine under the Oracle VM 2.2 environment. See http://download.oracle.com/docs/cd/E15458_01/doc.22/e15443/toc.htm.

Oracle Solaris 10 runs as a hardware virtual machine (HVM), which requires HVM support (Intel VT) on the underlying hardware platform—Oracle Solaris 10 has the paravirtualized (PV) drivers as part of the OS installed by default.

You need to check if the server has the HVM support. If you know the specific CPU model, you can find out if it supports HVM from Intel web site. You may need to modify the system BIOS setting to enable the HVM capabilities. If you already have Oracle VM 2.2 server installed, you can run the `xm info` command to verify if HVM is enabled. For example:

```
# xm info
release : 2.6.18-128.2.1.4.13.el5xen
virt_caps : hvm
xen_major : 3
xen_minor : 4
xen_extra : .0
xen_caps : xen-3.0-x86_64 xen-3.0-x86_32p hvm-3.0-x86_32 hvm-3.0-x86_32p hvm-3.0-
x86_64
```

For more information, see blogs.oracle.com/virtualization/.

D. Oracle Solaris Studio Tools Optimizations

There are several Oracle Solaris Studio tools optimizations.

Performance Counters on Oracle Solaris

Modern processors provide the ability to observe performance characteristics of applications using Performance Counters. For example, counters can be used to determine the average cycles per instruction for a given workload, determine how cache/memory intensive an application is, or determine whether there are any serious memory alignment issues with the way that an application lays out its data.

The Oracle Solaris dynamic tracing (DTrace) CPU Performance Counter provider (cpc provider) makes available probes associated with processor performance counter events. The cpc provider provides the ability to profile your application by many different types of processor related events, such as cycles executed, instructions executed, cache misses, TLB misses and many more.

In addition to DTrace, Oracle Solaris provides the `libcpc(3LIB)` API to access these performance counters. These interfaces can be used to observe the performance characteristics of applications.

Oracle Solaris also provides several utilities built upon `libcpc(3LIB)`) to make application analysis easier.

`cpustat(1M)`) allows you to observe performance characteristics on a system-wide level. Use the `-h` option to get a listing of all the different events that are available on a given processor. It is also possible to provide a raw event code instead of specifying the event name. This enables the case where it may not be obvious from the event name what event is being referred to, and for the case where a particular event name is not listed.

Note that `cpustat -h` does not list all possible combinations of umask for a given event, and you may need to be explicit.

`cputrack(1)` is used to analyze performance characteristics on a per-process or per-LWP basis.

The Oracle Solaris Studio Performance Analyzer, `analyzer(1)`, provides further functionality built on top of `libcpc(3LIB)`. The Analyzer uses the performance counters to help assess application performance and identify regions of code where performance problems occur. Analyzer is used to tune commercial applications and benchmarks to maximize platform-specific performance.

Modern Intel processors adhere to the Architectural Performance Monitoring specification. This means that means that going forward certain events, where available, are going to have consistent meanings across microarchitectures. Intel processors currently support Architectural Performance Monitoring versions 1 to 3. The current list of pre-defined Architectural Performance Monitoring events are:

- Unhalted Core Cycles
- Instructions Retired
- Unhalted Reference Cycles

- Last Level Cache References
- Last Level Cache Misses
- Branch Instructions Retired
- Branch Misses Retired

Any given processor may or may not support each of these events but when they are supported, they can be expected to be consistent across processors.

The availability of these events is programmatically determined by examining the Architectural Performance Monitoring Leaf `(0xA)` of the `CPUID` instruction.

Besides these pre-defined Architectural Performance Monitoring events, processors support a significant number of processor-specific performance monitoring events. These are listed in Appendix A of the *Intel 64 and IA-32 Architectures Software Developer's Manual*: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html.

Performance Counter Support on the Intel Xeon Processor

The Intel Xeon processor series has support for seven performance monitoring counters. Three of them are fixed function counters: `instr_retired.any`, `cpu_clk_unhalted.core`, and `cpu_clk_unhalted.ref`. The remaining four are programmable, allowing the user to measure any combination of the hundreds of available metrics.

Oracle Solaris `libcpc`(3LIB) on the Intel Xeon processor E7 family supports a full set of event names that are specific enough to define both the event code and unit mask (umask) value. In most cases, just the event name provided by `cpustat -h` is enough; there is no need to specify the umask explicitly. For additional, explicit control, event code and umask can be specified as numeric values.

In addition, the Intel Xeon processor E7 family provides counters that are 48-bits wide, up from 40-bit counters on older processors, enabling counting with lower overhead and fewer overflows.

The four programmable counters on the Intel Xeon processor E7 family can be configured to count a wide variety of events. For example, with the NUMA memory model on the Intel Xeon processor E7 family, memory placement (local or remote) can be an important contributor to the performance of an application. The Intel Xeon processor E7 family provides observability of this with the `mem_uncore_retired` series of events on the programmable counters.

Appendix A of the *Intel Software Developer's Manual Volume 3B* provides a description for how the following events count.

`mem_uncore_retired.other_core_l2_hitm`,
`mem_uncore_retired.remote_cache_local_home_hit`,
`mem_uncore_retired.remote_dram`, and
`mem_uncore_retired.local_dram`

In addition, the Intel Xeon processor E7 familysupport Architectural Performance Monitoring Version 3, which includes the `anythr` event attribute. This allows the user to count events for all logical processors on a core.

Useful Documentation

- Man pages for `cpustat(1M)`, `cputrack(1)`, `libcpc(3LIB)`, `cpc(3CPC)`, `collect(1)`, `analyzer(1)`, `er_print(1)` and `cpuid(7D)`
- Oracle Solaris Studio: http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html
- Chapter 30, "Performance Monitoring"of *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*: download.intel.com/design/processor/manuals/253669.pdf

E. Utilizing New Processor Instructions with Oracle Solaris

Modern processors provide instruction set extensions like the Streaming SIMD Extensions (SSE) available on the Intel Xeon processor E7 family. Utilizing these new instructions can improve performance of applications like speech, video and text processing.

SSE4.1 has more than 40 instructions mainly for packed data operations like `ROUNDPS` for rounding, `PMINSB` and `PMAXSB` for comparison, `PBLENDW` for blending, `PMULLD` for multiplication, `DPPS` for dot product, `MOVNTDQA` for nontemporal streaming loads, and `EXTRACTPS` for moving data between general-purpose registers and the `XMM` registers.

SSE4.2 includes the CRC32 instruction for error checking operations as well as instructions like `PCMPESTRM` and `PCMPESTRI` for string compare operations. For example, the Oracle Solaris iSCSI driver has been updated to utilize CRC32.

`POPCNT` counts the number of 1s in an operand.

The Oracle Solaris Cryptographic Framework uses AES-NI and `PCLMULQDQ` instructions in the cryptographic services it provides through kernel modules, user-level PKCS#11 API, and commands. The `PCLMULQDQ` instruction performs carry-less multiplication of two 64-bit quadwords, which are selected from the first and the second operands according to the immediate byte value.

Users of Oracle Solaris can determine which of these extensions are available on their system using the `isainfo(1)` command.

When an object is compiled, the hardware capabilities required for the object are stored in the object. The object is loaded at runtime only if the capabilities it requires are available on the machine. By failing to start the application, we avoid the scenario where the application faults when manipulating critical data and trying to execute an unsupported instruction. Use the `file(1)` or `elfdump(1)` utilities to see what features an application uses. Use the `elfedit(1)` utility to modify the hardware capabilities associated with an object.

Oracle Solaris also provides support for disassembly of these new instructions through `dis(1)`.

Useful Documentation

- Man pages for `dis(1)`, `isainfo(1)`, `file(1)`, `elfdump(1)`, `elfedit(1)`, `pargs(1)`, `cpuid(7D)`
- *Linker and Libraries Guide*, Oracle Solaris 10 Software Developer Collection: http://download.oracle.com/docs/cd/E19253-01/index.html
- Intel 64 and IA-32 Architectures Software Developer's Manuals: www.intel.com/products/processor/manuals/

Improved Microvectorization and Automatic Parallelization

The Intel Xeon processor E7 family provides 128-bit Steaming SIMD Extensions (SSE) instructions. These instructions perform integer and floating-point operations on vector operands. Utilizing these instructions can improve performance of media and scientific programs. To utilize the SSE instructions, developers typically write the assembly code manually or use the corresponding intrinsic calls in their program.

A better choice is using a compiler to utilize the SSE instructions automatically. Microvectorization is a compiler technology for this purpose. With this technology, compiled programs can achieve better performance automatically, without any source code modification. To enable this feature, Oracle Solaris Studio users need to specify `-xvector=simd` in their compiler options.

In Oracle Solaris Studio, microvectorization is applied on loops whenever safe. It can pack several iterations into one iteration, depending on the type of vectorized operations. For example, if there are single precise floating-point operations in a loop, microvectorization will pack four iterations into one iteration and use vector operations in the iteration.

Multiple core and multiple threads per core are increasingly an industry trend. The Intel Xeon processor E7 family feature Intel Hyper-Threading Technology, which provides two hardware threads per core and is capable of executing threads in parallel within each processor core.

To speed up serial application performance on such multi-threaded processors, one option is to use automatic parallelization. The compiler analyzes the programs, especially loops, and decides which loops can be safely parallelized. The compiler will then generate codes which, when executed at runtime, will have more than one thread to execute the loop body.

Two version codes for the loop may be generated if the compiler does not know the loop trip count; the parallel version of the code is executed at runtime only when the trip count is big enough.

To enable automatic parallelization, Oracle Solaris Studio users need to specify `-xautopar` in their compiler options. To enable parallelization of loops containing reduction operations, users need to specify `-xreduction` in their compiler options. A single one-line-per-loop summary of parallelization can be seen with option `-xloopinfo`. Users who want more detailed information should use compiler commentary.

As an important compiler optimization on the Intel Xeon processor E7 family, microvectorization is improved continually. Here are some new improvements in Oracle Solaris Studio.

- Vector expressions are subject to all optimizations applied to scalar ones, like Common Sub-expression Elimination (CSE), Dead Code Elimination (DCE), and so on.
- Improved loop transformation allows safe vectorization of more loops.
- More operations and function calls can be vectorized.
- Non-unit-stride loads and stores can be vectorized, depending on the well-tuned cost model.
- The range of loops that can be parallelized has increased by improvements in the compiler's data dependency analysis, ability to parallelize loops with wrap around data dependencies, and the ability to perform array reductions.
- Other general code transformations, like code specialization and loop fusion, have been improved to enable more profitable loop parallelization.

Useful Documentation

- *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1*: *Basic Architecture*, Chapter 10 through Chapter 12: www.intel.com/Assets/PDF/manual/253665.pdf
- Oracle Solaris Studio 12.2 documentation: http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/oss-122-docs-169726.html

Instruction Selection

The new Intel Xeon processors E7 family has differences from previous 64-bit processors that the compiler can exploit in the way it generates code. Previous processors had floating-point functional units that were 64-bits wide. That is, even though they were representing `%xmm` registers that were 128 bits wide, their internal registers and floating point units were 64-bits wide. Operations were split into two pieces by the microcode, and processed as two 64-bit pieces. This is one of the reasons it is harder to get the older 64-bit processors to perform well with 128-bit SIMD instructions. The operations were split into two 64-bit operations by the microcode, so performance is not that much different than just using two 64-bit instructions. For this reason, on older processors, you will see the compiler generating pairs of `movlpd` and `movhpd` instructions to load two parts of a 128-bit operands to act more like the 64-bit internal microcode, which enables a better pipelining opportunities. For example:

```
movlpd variable, %xmm0
movhpd variable+8, %xmm0
```

Instead of:

```
movupd variable, %xmm0
```

This typically only happens for unaligned data. Aligned data is better processed with `movapd` instructions because the microcode deals very well with aligned data and streaming operations. For 64-bit scalar operations, a single `movlpd` instruction is used in favor of a `movsd` instruction, which doesn't waste time zeroing out the high end of the `%xmm` register. 128-bit SIMD operations are still better in applications that are very stream oriented, such as BLAS libraries.

The newer Intel Xeon processor E7 family have functional units that are a full 128- bits wide with 128-bit internal registers. That is good for SIMD operations because the microcode no longer splits the 128-bit operation into two 64-bit operations, so bandwidth is improved. However, other issues can arise. Since the internal register is now 128-bits wide, if two 64-bit operands are being loaded into the high and low parts of the 128-bit register, care must be taken on how that is done; otherwise, performance suffers due to something called a partial register stall. To load a value in the high part of the `%xmm` register, the internal microcode register must be loaded, the high part modified, and then the register is stored back. If that internal microcode register is still waiting on the low part to be modified, a stall can happen. So, the old method of generating `movlpd` and `movhpd` instructions does not work very well on new Intel Xeon processor E7 family because the `movhpd` can stall waiting for the `movlpd` instruction to complete. A better solution is to load the first 64-bit operand into the lower part of the `%xmm` register, zeroing the high part with a `movsd` or `movq` instruction. This modifies the entire 128-bit `%xmm` register. Then follow that load instruction with a `movhpd` to load the second 64-bit operand into the high part of the `%xmm` register. In this case, the microcode, knowing that the `%xmm` is being cleared in the high part, will not stall waiting for completion of the low part. For example:

```
movsd variable, %xmm0
movhpd variable+8, %xmm0
```

Or

```
movq variable, %xmm0
movhpd variable+8, %xmm0
```

This problem is not limited to load pairs for SIMD operations. It can also happen in scalar code when using the `movlpd` instruction to load the 64-bit operands into the low part of a `%xmm` register. Because the `movlpd` instruction does not modify the high 64 bits, it has to merge the 64-bit operand into the lower part of the `%xmm` register, preserving the high part and causing a stall. A single `movsd` or `movq` will zero the high part, so entire register can be written at once and no stall will occur. The overall performance difference in floating point intensive code can be as much as 25 percent when the correct load instructions are used.

Store instructions are handled differently than load instructions because you can read the high and low parts of a `%xmm` register without causing a stall. Also, feeding some store units 64 bits at a time works better.

	-M64 GENERIC32/64	-XCHIP=CORE2 -XCHIP=PENRYN WOODCREST32/64 PENRYN32/64	-XCHIP=PENTIUM4 -XCHIP=NEHALEM PENTIUM4 NEHALEM32/64
Scalar Reg Move Float	movaps	movaps	movaps
Scalar Load Float	movss	movss	movss
Scalar Store Float	movss	movss	movss
Scalar Reg Move Double	movapd	movapd	movapd
Scalar Load Double	movsd	movsd	movsd
Scalar Store Double	movsd	movsd	movsd
SIMD Reg Move Float	movaps	movaps	movaps
SIMD Reg Move Double	movapd	movapd	movapd
Unaligned Load Float	movups	movq/movhps	movups
Unaligned Store Float	movlps/movhps	movlps/movhps	movups
Unaligned Load Double	movupd	movsd/movhpd	movupd
Unaligned Store Double	movlpd/movhpd	movsd/movhpd	movupd
Unaligned Load Quad	movdqu	movq/movhpd	movdqu
Unaligned Store Quad	movlpd/movhpd	movq/movhpd	movdqu
Aligned Load Float	movaps	movaps	movaps
Aligned Store Float	movaps	movaps	movaps
Aligned Load Double	movapd	movapd	movapd
Aligned Store Double	movapd	movapd	movapd
Aligned Load Quad	movdqa	movdqa	movdqa
Aligned Store Quad	movdqa	movdqa	movdqa

The Intel Xeon processor E7 family deserves special mention because Intel has improved the microcode to deal with many of these issues. In the Intel Xeon processor E7 family, the compiler will generate much more generic code. An unaligned 128-bit load is not that much slower than an aligned 128-bit load and it does not have to be split into two instructions.

The table below shows the current load/store/move combinations for float and double data types, for scalar and SIMD, both aligned and unaligned, that the compiler generates. Always choose the processor that you are running on when compiling your application for maximum performance. The `-fast` or `-native` compiler flags will make that choice automatically.

Note that if you want to run your application on multiple architectures, it gets more complicated. If they are all Core2 processors, choosing `-xchip=core2` may be the best choice because the Intel Xeon processor E7 familywill execute the Core2/Penryn load/store combinations quite well. If there are Pentium®4, Core2 and Intel Xeon processor E7 familyin the mix, then `-xchip=generic` may be the best choice. In all cases, you really should try different `-xchip` choices and check your application's performance.

The compiler also must be careful with generating code for arithmetic conversion operations on the new Intel Xeon processor E7 family. The same issues described earlier regarding partial register stalls come into play. That is, writing into only part of a 128-bit `%xmm` register can cause a stall while the microcode reads the internal register, modifies only part of it and then writes it back. If that register is involved with some other operation, then the microcode may have to wait for this merge operation to finish. To avoid these issues, the compiler will generate code to write the entire 128-bit register when it can. It must be careful to know that the upper part of the register for scalar operations has been zeroed. Also, some processors can benefit by clearing a `%xmm` register with an `xorps` instruction to clear the contents of the entire register just before writing into it with a convert instruction. It works much in the same way as using `movsd/movhpd` combinations to load the low part of a register and zero to high part just before loading another operand into the high part. The microcode notices the `xorps` instruction has cleared the entire contents of the target register or the convert operation. Knowing that, the microcode can write the `convert` instruction results into that cleared register without waiting on some other operation to complete.

INSTRUCTIONS GENERATED BY THE COMPILER FOR DIFFERENT DATA TYPE CONVERSIONS AND DIFFERENT `-XCHIP` SELECTIONS.

OPERATION	SRC	WOODCREST	PENRYN	NEHALEM	GENERIC/PENTIUM4
int to double	mem	movdl	movdl	xorps	cvtsi2sd mem,xmm
		mem,xmm	mem,xmm	xmm,xmm	
		cvtdq2pd	cvtdp2pd	cvtsi2sd	
		xmm,xmm	xmm,xmm	mem,xmm	
	reg	movdl	movdl	xorps	cvtsi2sd gpr,xmm
		gpr,xmm	gpr,xmm	xmm,xmm	
		cvtdq2pd	cvtdq2pd	cvtsi2sd	
		xmm,xmm	xmm,xmm	gpr,xmm	
long to double	mem	cvtsi2sdq	xorps	xorps	cvtsi2sdq mem,xmm
		mem,xmm	xmm,xmm	xmm,xmm	
			cvtsi2sdq	cvtsi2sdq	
			mem,xmm	mem,xmm	
	reg	cvtsi2sdq	xorps	xorps	cvtsi2sdq
		gpr,xmm	xmm,xmm	xmm,xmm	gpr,xmm
			cvtsi2sdq	cvtsi2sdq	
			gpr,xmm	gpr,xmm	
int to float	mem	movdl	movdl	movdl	movdl mem,xmm
		mem,xmm	mem,xmm	mem,xmm	
		cvtdq2ps	cvtdq2ps	cvtdq2ps	cvtdq2ps xmm,xmm
		xmm,xmm	xmm,xmm	xmm,xmm	
	reg	movdl	movdl	movdl	movdl gpr,xmm
		gpr,xmm	gpr,xmm	gpr,xmm	
		cvtdq2ps	cvtdq2ps	cvtdq2ps	cvtdq2ps
		xmm,xmm	xmm,xmm	xmm,xmm	xmm,xmm
long to float	mem	xorps	xorps	xorps	xorps xmm,xmm
		xmm,xmm	xmm,xmm	xmm,xmm	
		cvtsi2ssq	cvtsi2ssq	cvtsi2ssq	cvtsi2ssq
		mem,xmm	mem,xmm	mem,xmm	mem,xmm
	reg	xorps	xorps	xorps	xorps
		xmm,xmm	xmm,xmm	xmm,xmm	xmm,xmm
		cvtsi2ssq	cvtsi2ssq	cvtsi2ssq	cvtsi2ssq
		gpr,xmm	gpr,xmm	gpr,xmm	gpr,xmm
double to float	mem	movsd	movsd	movsd	xorps xmm,xmm
		mem,xmm	mem,xmm	mem,xmm	
		cvtpd2ps	cvtpd2ps	cvtpd2ps	cvtsd2ss mem,xmm
		xmm,xmm	xmm,xmm	xmm,xmm	
	reg	cvtsd2ss	cvtsd2ss	cvtsd2ss	xorps xmm,xmm
		xmm,xmm	xmm,xmm	xmm,xmm	
					cvtsd2ss xmm,xmm
float to double	mem	movss	movss	movss	cvtss2sd mem,xmm
		mem,xmm	mem,xmm	mem,xmm	
		cvtps2pd	cvtps2pd	cvtps2pd	
		xmm,xmm	xmm,xmm	xmm,xmm	
	reg	cvtss2sd	xorps	xorps	cvtss2sd xmm,xmm
		xmm,xmm	xmm,xmm	xmm,xmm	
			cvtss2sd	cvtss2sd	
			xmm,xmm	xmm,xmm	

Note that there are other cases where certain instructions were avoided on older 64-bit processors. `shufpd` and `shufps` were good examples. These were expensive instructions, and combinations of `movhps`, `movhpd` and `unpckhpd` were used instead of a `shufpd`. The Intel Xeon processor E7 family no longer have this problem and the `shufpd` and `shufps` instructions are now being used more often. Overall, any time you are looking for maximum performance, selecting the `-xchip` option for the processor where the application will run is very important.

OpenMP Development with Oracle Solaris Studio

The Intel Xeon processor E7 family features Hyper-Threading Technology, which provides two hardware threads per processor core and is capable of executing threads in parallel within each core. So an ten-core Intel Xeon processor provides a total of 20 hardware threads.

To take advantage of the parallelism offered by these processors, one needs to write multithreaded programs. In a multithreaded program, multiple threads work together within the context of a single process, thus speeding up the execution of the program.

In what follows, we introduce the OpenMP API, a popular standard for writing multithreaded programs.

The OpenMP API

The OpenMP Application Programming Interface (API) enables users to develop multithreaded, shared-memory programs in C, C++, and Fortran. The advantages of using the OpenMP API are performance, scalability, portability, and standardization. With a relatively small amount of coding effort, a programmer can write multithreaded applications to run on a multicore machine.

The OpenMP API is composed of three components:

- Compiler directives
- Runtime library routines
- Environment variables

Compiler directives are instructions to the compiler. Starting with a sequential program, the programmer can incrementally insert OpenMP directives in the code that instruct the compiler how to parallelize the program.

In C and C++, OpenMP directives are specified using the `#pragma omp` mechanism. In Fortran, OpenMP directives are specified using special comments that are identified by the unique sentinels `!$omp`, `c$omp`, or `*$omp`.

The OpenMP API has a rich set of directives that the programmer can use to specify parallelism in a program. The essential directive for creating threads is the `parallel` directive. The `parallel` directive specifies that a region of code should be executed in parallel by multiple threads.

An OpenMP program begins as a single thread of execution, called the initial thread. When a thread encounters a `parallel` construct, it creates a new team of threads composed of itself and zero or more additional threads, and becomes the master of the team. All members of the team (including the master) execute the code inside the parallel construct. There is an implicit barrier at the end of the `parallel` construct. Only the master thread continues execution of the code beyond the end of the `parallel` construct.

The number of threads used to execute a parallel region can be specified by using the `num_threads` clause on the `parallel` directive, by calling the `omp_set_num_threads()` runtime library routine, or by setting the environment variable `OMP_NUM_THREADS`.

Example 1

In the following C/C++ program, there is a parallel region that will be executed by four threads. Each of the four threads executes the code in the region concurrently and prints "Hello world."

```
#include <stdio.h>
#include <omp.h>

void main(void)
{
  omp_set_dynamic(0);
  omp_set_num_threads(4);

#pragma omp parallel
  {
    printf ("Hello world\n");
  }
}
```

The output of the above program will look as follows.

```
Hello world
Hello world
Hello world
Hello world
```

The OpenMP programming model is the shared-memory programming model. Variables in a parallel region are shared by default. OpenMP data-sharing attribute clauses allow the programmer to control the data-sharing attributes of variables. The clauses include the `shared`, `private`, `firstprivate`, `lastprivate`, and `reduction` clauses.

Example 2

The following C/C++ program illustrates the use of the `shared` and `private` clauses. The `shared` clause on the `parallel` directive specifies that the variable `x` is shared, so all threads executing the parallel region will reference the same copy of `x`. The `private` clause specifies that the variable `id` is private, so each thread executing the parallel region will reference its own private copy of `id`.

While executing the parallel region, a thread stores its OpenMP thread number in its copy of `id` and prints the value of that copy. The thread then increments `x` by 1 inside a critical region denoted by the `critical` directive. A critical region is used to avoid data race conditions since all the threads reference the same copy of `x`. At the end of the parallel region, only the master (initial) thread continues execution and prints the value of `x`.

```
#include <stdio.h>
#include <omp.h>

void main(void)
{
  int x, id;

  omp_set_dynamic(0);
  omp_set_num_threads(4);

  x = 0;

#pragma omp parallel shared(x) private(id)
  {
    id = omp_get_thread_num();
    printf ("My thread id is %d\n", id);

    #pragma omp critical
      {
         x = x + 1;
      }
  }
  printf ("After parallel region, x = %d\n", x);
}
```

The output of the above program may look as follows. Note that since the threads are running in parallel, the order in which the lines are printed may vary from run to run.

```
My thread id is 0
My thread id is 3
My thread id is 2
My thread id is 1
After parallel region, x = 4
```

OpenMP Specification Version 3.0

The OpenMP specification is the definitive reference on the OpenMP API. The specification is owned and managed by the OpenMP Architecture Review Board (ARB), a non-profit organization established in 1997.

The latest OpenMP specification is version 3.0, which was released in May 2008. The 3.0 specification includes the following new features:

- The `task` directive, which facilitates the parallelization of codes where the units of work are generated dynamically (as in a recursive structure or a while loop)
- The `collapse` clause, which instructs the compiler to collapse perfectly nested loops and then parallelize the resulting loop
- Better control over nested parallel regions, and new OpenMP routines to determine nesting structure

Example 3

The following recursive C/C++ program shows how the OpenMP `task` directive can be used to compute the number `Fn` (for `n=20`) in the Fibonacci sequence. In the program, there is a parallel region that is executed by a team of 8 threads. The 8 threads will execute all the tasks that are generated in the parallel region. The call to `fib(n)` in the `parallel` region is enclosed by the `master` directive, so only the master thread will execute the call to `fib(n)` (for `n=20`), while the other threads will continue to the (implicit) barrier at the end of the parallel region.

While executing `fib()`, the master thread encounters two `task` constructs and generates two tasks. One of the tasks calls `fib(n-1)` and the other calls `fib(n-2)`, and the return values of these calls are added together to produce the value returned by `fib(n)`. Each of the calls to `fib(n-1)` and `fib(n-2)` will in turn generate two tasks. Tasks will be recursively generated until the argument passed to `fib()` is less than 2. Any of the threads in the team may execute the generated tasks. When all the tasks generated have been executed, the parallel region ends. After the parallel region, only the master (initial) thread will continue execution and print the result.

The `taskwait` directive ensures that the two tasks generated in an invocation of `fib()` are completed (that is. the tasks compute `i` and `j`) before that invocation of `fib()` returns.

Note that although only the master thread executes the `master` directive and hence the first call to `fib()`, all 8 threads will participate in executing the tasks generated.

```
#include <stdio.h>
#include <omp.h>

int fib(int n)
{
  int i, j;
  if (n<2)
    return n;
  else
    {
      #pragma omp task shared(i) firstprivate(n)
      i=fib(n-1);
```

```
      #pragma omp task shared(j) firstprivate(n)
      j=fib(n-2);

      #pragma omp taskwait
      return i+j;
    }
}

int main()
{
  int n = 20;
  int result;

  omp_set_dynamic(0);
  omp_set_num_threads(8);

  #pragma omp parallel shared(n, result)
  {
     #pragma omp single
      result = fib(n);
  }

  printf ("fib(%d) = %d\n", n, result);
}
```

The output of the above program will look as follows.

```
fib(20) = 6765
```

Oracle Solaris Studio

The Oracle Solaris Studio software is a comprehensive, integrated set of C, C++, and Fortran compilers and tools that enable the development and deployment of applications on a variety of platforms, including the Intel Xeon processor E7 family. Oracle Solaris Studio software is available for free.

The Oracle Solaris Studio compilers fully support OpenMP specification version 3.0. When compiling an OpenMP program, specify the `-xopenmp` compiler option. This option enables the compiler to recognize OpenMP directives and transform the code so it can run using multiple threads.

In addition to the compilers, Oracle Solaris Studio includes a variety of tools that facilitate OpenMP programming. Some of these tools are described below.

OpenMP Debugging

The Oracle Solaris Studio dbx tool can be used to debug OpenMP programs. An OpenMP program should first be prepared for debugging with dbx by compiling it with the options

```
-xopenmp=noopt -g.
```

All of the dbx commands that operate on threads can be used for OpenMP debugging. dbx allows the user to single-step into a `parallel` region, set breakpoints in the body of an OpenMP construct, as well as print the values of `shared`, `private`, `threadprivate`, as well as other variables for a given thread.

Data Race and Deadlock Detection

The Oracle Solaris Studio Thread Analyzer tool helps the programmer detect data races and deadlocks in an OpenMP program.

A data race occurs when:

- Two or more threads in a single process access the same memory location concurrently,
- At least one of the accesses is a write, and
- The threads are not holding any exclusive locks to control their accesses to that memory location.

When these three conditions hold, the order of accesses to the memory location is non-deterministic, and the computation may give different results from one run to another depending on the order. Data races in a program are notoriously hard to detect.

A deadlock occurs when:

- Threads that are already holding locks request new locks,
- The requests for new locks are made concurrently, and
- Two or more threads form a circular chain in which each thread waits for a lock that is held by the next thread in the chain.

When these three conditions hold, two or more threads are blocked (hung) forever because they are waiting for each other. The advantage of the Thread Analyzer is that it can detect potential deadlocks that may not have occurred in a particular run, but may occur in other runs.

OpenMP Performance Analysis

The Collector and the Performance Analyzer are a pair of tools in Oracle Solaris Studio that can be used to collect and analyze performance data for an application. The Collector tool collects performance data using a statistical method called profiling and by tracing function calls. The Performance Analyzer processes the data recorded by the Collector, and displays various metrics of performance at program, function, OpenMP parallel region, OpenMP task region, source-line, and assembly instruction levels. The Performance Analyzer can also display the raw data in a graphical format as a function of time.

Summary

The OpenMP API is the de-facto standard for writing multi-threaded applications to run on multi-core machines. The OpenMP specification version 3.0 defines a rich set of directives, runtime routines, and environment variable that allow the programmer to write multi-threaded applications in C, C++, and Fortran.

Oracle Solaris Studio facilitates OpenMP development. Oracle Solaris Studio compilers (C, C++, and Fortran) support the OpenMP specification version 3.0. Various Oracle Solaris Studio tools aid in the debugging, error checking, and performance analysis of OpenMP programs.

Useful Documentation

- Official OpenMP Architecture Review Board Web page: www.openmp.org
- OpenMP Application Program Interface specification version 3.0: www.openmp.org/mp-documents/spec30.pdf
- Oracle Solaris Studio: http://www.oracle.com/technetwork/server-storage/solarisstudio/overview/index.html
- Oracle Solaris Studio OpenMP information (midway down the page): http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/programming-jsp-139962.html

Oracle Solaris Operating System: Optimized for Sun x86 Systems in the Enterprise
August 2011, Version 1.1

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

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

Hardware and Software, Engineered to Work Together