THE SOLARIS™ OPERATING SYSTEM—OPTIMIZED FOR THE INTEL® XEON® PROCESSOR 5500 SERIES

Jonathan Chew, Sun Microsystems, Inc,
Kuriakose Kuruvilla, Sun Microsystems, Inc.
Eric Saxe, Sun Microsystems, Inc.
Rafael Vanoni, Sun Microsystems, Inc.
Sherry Moore & Juanita Heieck, Sun Microsystems, Inc.
Yi Gao & Yonghong Song, Sun Microsystems, Inc.
David Seberger, Sun Microsystems, Inc.
Nawal Copty, Sun Microsystems, Inc.

Sun BluePrints™Online

Part No. 820-7707-**1**
Revision 1.1, 03/30/09

Table of Contents

Introduction **1**
Background 1
The Solaris Ecosystem 2

Intelligent Performance **3**
Overview 3
NUMA and Multi-threading 3
Performance Counters on Solaris 4

Automated Energy Efficiency **7**
Power Aware Dispatcher 7
PowerTOP for OpenSolaris 8

Reliability and Availability **10**
Fast Reboot 10
Support for Microcode Updates on Intel Processors 13

Developer Tools **15**
Utilizing New Processor Instructions with the Solaris OS 15
Improved Microvectorization and Automatic Parallelization 16
Instruction Selection 18
OpenMP Development with Sun Studio 22

Appendices **26**
About the Authors 26
Acknowledgements 27
Ordering Sun Documents 27
Accessing Sun Documentation Online 27

Chapter 1

Introduction

This document is intended as a quick reference guide for developers and system administrators that want to optimize the Solaris™ OS on the Intel® Xeon® processor 5500 series platform. The paper includes a short overview of the Sun and Intel collaboration, and brief technical descriptions of specific features and capabilities that can be implemented in the Solaris OS to optimize the specific capabilities of Intel Xeon processor 5500 series- systems, specifically in the areas of:

- Intelligent performance, on page 3
- Automated energy efficiency, on page 7
- Reliability and availability, on page 10
- Developer tools, on page 15

Background

Sun and Intel, as part of a broad strategic alliance, have been working together—from design and architecture through implementation—to ensure that the Solaris OS is optimized to unleash the power and capabilities of current and future Intel Xeon processors at the time of launch. Sun and Intel have made significant advances to optimize the Solaris OS for Intel Xeon processor-based systems, and are working closely to unleash new capabilities that are part of the Intel Xeon processor 5500 series. Engineering teams from the two companies are collaborating on optimizing how the Solaris ecosystem and the Intel Xeon processor 5500 series work together, with compelling results, including:

- Increased performance as the Solaris OS takes advantage of Intel® multi-core processor capabilities and Intel® Turbo Boost Technology.
- Optimized power efficiency and utilization by enabling Solaris to take advantage of Intel Xeon processor 5500 series performance-enhanced dynamic power management capabilities.
- Extending Predictive capabilities to improve reliability by incorporating Intel Xeon processor 5500 series features into the Solaris Fault Management Architecture (FMA).

The Solaris Ecosystem

The Solaris ecosystem consists of the Solaris OS and open source OpenSolaris™ OS, as well as the Sun Studio development tools, which form the core of a large developer community and a vast portfolio of applications. The free and open Solaris OS is a proven, industry leading operating system with features designed to save time and money in business-critical operations. The Solaris OS provides stability, massive scalability, high performance, and guaranteed forward binary compatibility. Intel is embracing Solaris as a mainstream OS and the enterprise class, mission critical UNIX® OS for Intel Xeon processor-based servers.

The OpenSolaris community is where the next generation of Solaris is being built, and where the latest innovations from Sun and Intel can be found. The OpenSolaris OS offers cutting-edge features contributed by a global development community, which provides accelerated time to market and support for the latest technologies and innovation in an environment familiar to GNU/Linux and UNIX developers and administrators. Future releases of the Solaris OS will be based on OpenSolaris OS. Solaris releases feature an extended support life cycle as needed in today's demanding datacenters. More information can be found at sun.com/solaris.

The Solaris ecosystem, deployed with the Intel Xeon processor 5500 series, will be an outstanding choice for both leading edge applications such as Web 2.0 and high performance computing, and all forms of enterprise computing. Organizations can leverage Solaris as the mission-critical enterprise class operating system on Intel Xeon processor 5500 series systems from Sun, as well as other manufacturers—the Solaris OS is supported on over 1,100 systems, and OpenSolaris on over 4,000 systems.

Chapter 2

Intelligent Performance

Overview

Building on a proven track record, the Solaris OS is ready to take advantage of the groundbreaking performance capabilities of the Intel Xeon processor 5500 series. Significant performance innovation comes from optimizations of the individual cores and the overall multi-core microarchitecture, which increase both single-threaded and multi-thread performance. As a result, the Solaris kernel and existing single- or multi-threaded applications will run faster, with no code changes or recompilation necessary.

Intel Turbo Boost Technology uses any available power headroom to deliver higher clock rates. In those situations where the application requires maximum processing power, the Intel Xeon processor 5500 series increases the frequency in the active core when conditions such as load, power consumption and temperature permit it.

The 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 new Intel Xeon processor 5500 series, the Solaris OS enables new levels of performance as applications incorporate multi-threaded design, to increase throughput, responsiveness, efficiency, scalability, and overall performance.

NUMA and Multi-threading

The Solaris OS takes advantage of the capabilities of the new Intel® QuickPath Interconnect (Intel® QPI) architecture with capabilities such as an optimized scheduler and memory placement optimization (MPO) capability that has proven performance benefits with non-uniform memory access (NUMA) architecture systems. This improves overall performance by reducing memory latency. The Solaris NUMA implementation takes information from the Advanced Configuration and Power Interface (ACPI) System Resource Affinity Table (SRAT) and the System Locality Information Table (SLIT).

- The SRAT stores topology information for all the processors and memory, describing their physical locations. The Solaris OS scans the ACPI SRAT on the Intel Xeon processor 5500 series at boot time and uses the information to better allocate memory and schedule software threads for maximum performance.
- The SLIT stores node-to-node latency information.

MPO leverages the NUMA topology using SLIT/SRAT features in the Intel Xeon processor 5500 series to maximize performance. Solaris has long supported MPO, for even the most complex NUMA topologies to ensure maximum performance in multi-processor systems.

Modern processors feature multiple processor cores in a single socket, with multiple hardware threads per core. On the Intel Xeon processor 5500 series, Intel® Hyper-Threading Technology offers two hardware threads, with shared execution pipeline and L1 cache. The Solaris OS is aware of Intel Hyper-Threading Technology, ensuring optimal scheduling of software threads to minimize resource contention and maximize performance.

Performance Counters on 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; or to determine how cache/memory intensive an application is; or if there are any serious memory alignment issues with the way that an application lays out its data.

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

Solaris also provides a few 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.

Performance Analyzer tools like `collect(1)`, `analyzer(1)` and `er_print(1)` provide further functionality built on top of `libcpc(3LIB)`.

The 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.

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 counting of a significant number of performance events. These are listed in Appendix A of *Intel 64 and IA-32 Architectures Software Developer's Manual,* specified in the Useful Documentation section.

Useful documentation

Man pages for `cpustat(1M)`, `cputrack(1)`, `libcpc(3LIB)`, `cpc(3CPC)`, `collect(1)`, `analyzer(1)`, `er_print(1)` and `cpuid(7D)`)

Sun Studio 12: Performance Analyzer http://docs.sun.com/app/docs/doc/819-5264

Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2 , available at *http://download.intel.com/design/processor/manuals/253669.pdf*

– See *Chapter 18 Debugging and Performance Monitoring and Appendix A Performance-Monitoring Events.*

Chapter 3

Automated Energy Efficiency

Power Aware Dispatcher

Solaris takes advantage of many power efficiency features in the Intel Xeon processor 5500 series. For example, an innovative Power Aware Dispatcher has been integrated into OpenSolaris, enabling Intel Xeon processor 5500 series to stay longer in idle states (C-states), have better granularity in power management (P-states), and better responsiveness to changes in demand. 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.

In OpenSolaris, the kernel dispatcher—the part of the kernel that decides where threads should run—will be integrated with the power management subsystem of the Intel Xeon processor 5500 series. The Solaris Power Aware Dispatcher (PAD) has increased awareness of the Intel Xeon processor 5500 series, such that the workload can be efficiently utilized on available hardware threads, with benefits for shared pipelines, shared caches, and shared sockets. The Solaris PAD is able to communicate what processor resources are being used by the operating system, and which are not. The Solaris kernel now has the ability to utilize those parts of the processor that are active, and continue to avoid doing work on those parts that are powered down.

Together, these two capabilities work together resulting in greater power efficiency without losing any performance. This integration enables the Intel Xeon processor 5500 series to enter into a deeper C-state, and do so with less latency. These capabilities are enabled by default.

Useful documentation

http://opensolaris.org/os/project/tesla/Work/CPUPM/

PowerTOP for OpenSolaris

Modern microprocessors have the ability to operate at different frequency and idle levels to reduce overall power consumption. These features are controlled by the operating system, which decides, based on overall system utilization, how much of the hardware's processing capabilities are needed. This method has proven successful as a way of saving power and therefore lowering operating costs.

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 the user to understand which applications that run on his 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. By fixing, filing bugs against, or just not running power inefficient software the user can help improve the system's power efficiency.

PowerTOP for OpenSolaris leverages the 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 are displayed at the top of the terminal, while a listing of the most recent and frequent events is displayed below.

By utilizing Solaris 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 the Intel Xeon processor 5500 series power management system. PowerTOP will be able to monitor and measure Turbo Boost.

PowerTOP was originally conceived by the Intel® Corporation for Linux based systems, and developed as an open source tool. Sun partnered with Intel to develop PowerTOP for OpenSolaris in the OpenSolaris community, resulting in the integration of PowerTOP into OpenSolaris 2008.11.

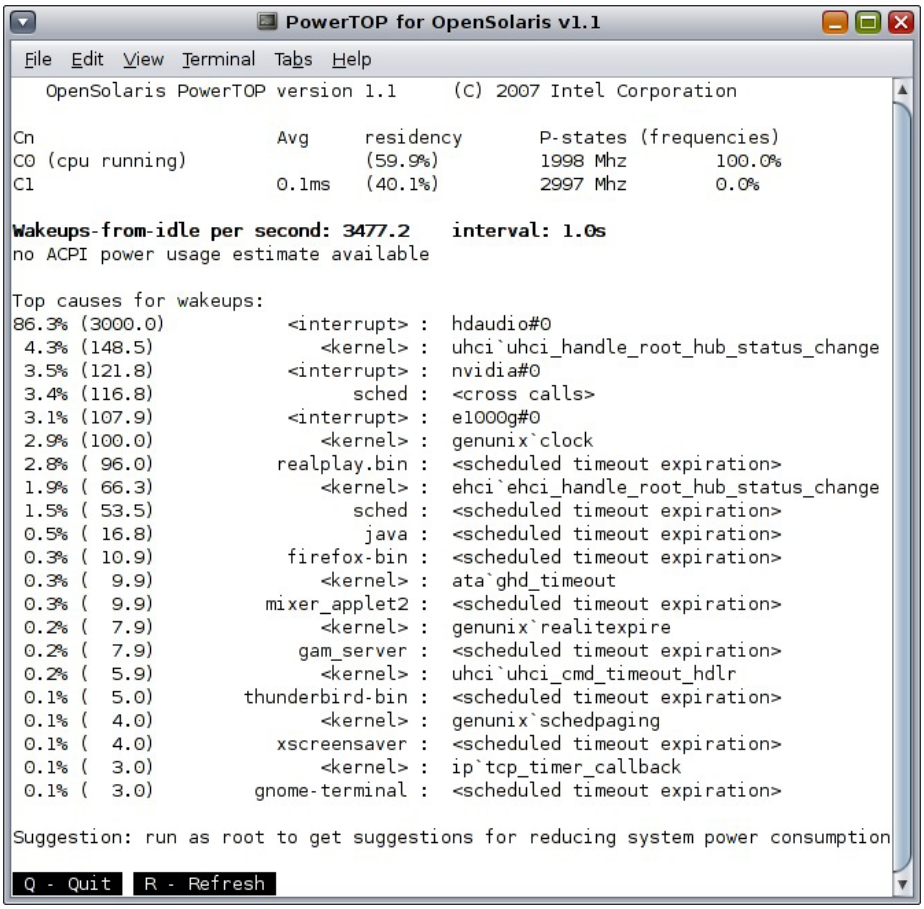

Figure 1: PowerTOP for OpenSolaris

Useful documentation

Man page for `powertop(1)`, PowerTOP's project page on opensolaris.org *(http://opensolaris.org/os/project/tesla/Work/Powertop)*

Chapter 4

Reliability and Availability

The Solaris Operating System provides a proven architecture for building and deploying systems and services capable of Predictive Self Healing, which automatically diagnoses, isolates, and aids in recovery from hardware and application faults. Solaris Fault Manager is a key component of Predictive Self Healing. Solaris Fault Manager receives data relating to hardware and software errors and automatically diagnoses the underlying problem. Once diagnosed, Solaris Fault Manager automatically responds by offlining faulty components. Sun and Intel are working together to extend these capabilities to systems based on the Intel Xeon processor 5500 series, including chipsets and memory subsystems. Solaris recognizes and supports Intel Xeon processor 5500 series Machine Check Architecture (MCA).

Sun and Intel are working together to leverage status information throughout the Intel Xeon processor 5500 series to help the Solaris fault management system diagnose a hardware fault correctly. This verification ensures that users running the Solaris OS on Intel Xeon processor-based systems will get a correct diagnosis and recovery should a hardware fault occur.

One critical element to availability is reliable data subsystems. Solaris ZFS™ provides unparalleled data integrity, capacity, performance, and manageability. Solaris 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. Solaris ZFS was introduced in the Solaris 10 6/06 update, and is the default file system in the current OpenSolaris OS release and all future Solaris releases.

Fast Reboot

The ability to fast reboot a system drastically reduces downtime and improves efficiency. Fast Reboot is a command-line feature that enables you to reboot an Intel Xeon processor 5500 series system quickly, bypassing the BIOS, power on self test, and GRUP Bootloader.

Fast Reboot implements an in-kernel boot loader that loads the kernel into memory and then switches to that kernel, so that the reboot process occurs within seconds. For example, internal testing has shown that system reboot time improves dramatically for a Sun Fire server configured with an Intel Xeon processor 5500 series.

This feature is implemented on both 32-bit and 64-bit kernels.

To support the Fast Reboot feature, the following new options have been added to the `reboot` command:

- `-f` Initiates the fast reboot process, when used with the `reboot` command.
- `-e` Initiates a fast reboot of the system to an alternate boot environment (BE), when used in conjunction with the reboot command and the `-f` option.

Note that the `-e` option cannot be used in the OpenSolaris 2008.11 release to initiate a fast reboot to an alternate BE. For instructions, see the product documentation at *http://docs.sun.com/app/docs/doc/819-2379/ghsqy?a=view*

The system's capability to bypass the firmware when booting a new OS image has dependencies on device drivers' implementation of a new device operation entry point, `quiesce`.

On supported drivers, this implementation quiesces a device, so that at completion of the function, the driver no longer generates interrupts or accesses memory. The quiesce function also resets the device to a hardware state, from which the device can be correctly configured by the driver's attach routine, without a power cycle of the system or being configured by the firmware. For more information, see the `quiesce(9E)` and `dev_ops(9S)` man pages.

To make a fast reboot the default behavior on your system, create a `fastreboot` file in the `/etc directory`.

```
# touch /etc/fastreboot
```

The addition of the `fastreboot` file on the system changes the behavior of the reboot command, so that it uses the `-f` option to initiate a fast reboot, by default.

To revert to the original behavior of the reboot command, remove the file.

```
# rm /etc/fastreboot
```

Note: Removing this file does not remove the fast reboot capability from the system.

There is an undocumented internal command to check whether a fast reboot attempt will succeed.

```
# reboot -f dryrun
```

On success:

```
# reboot -f dryrun
reboot: all drivers have implemented quiesce(9E)
```

On failure:

```
# reboot -f dryrun

(Drivers without quiesce() implementation will be listed)

 genunix: WARNING: nvidia has no quiesce()

 reboot: not all drivers have implemented quiesce(9E)
```

If you don't have console access, you can get the list of drivers without `quiesce(9E)` support like this:

```
# grep "no quiesce" /var/adm/messages
```

Other changes that support Fast Reboot on the Intel Xeon platform include the new `uadmin` function, `AD_FASTREBOOT`. This function resets the system, enabling the reboot command to bypass the `BIOS` and the boot loader phases.

Note: The `uadmin` command has limited functionality. If the command is used to facilitate a fast reboot of the system, neither the boot archive, nor the `menu.lst` file are updated. For this reason, the `reboot -f` command is the preferred method for initiating a fast reboot of the system.

Future Solaris releases will implement fast reboot capabilities, including a boot configuration service that enables fast reboot and panic fast reboot as the default behavior of the `reboot` command.

Useful documentation

http://wikis.sun.com/display/OpenSolarisInfo/Support+for+Fast+Reboot

http://wikis.sun.com/display/OpenSolarisInfo/Using+Fast+Reboot

http://dlc.sun.com/osol/docs/content/SYSADV1/ghsbc.html

http://blogs.sun.com/sherrym/date/20080922

Support for Microcode Updates on Intel Processors

The Intel family of processors have 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 the Solaris OS. 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 run time 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 the `/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, run time 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_drv` driver 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.

Chapter 5

Developer Tools

Optimizations made to the compiler tools and runtime libraries from both Sun and Intel help developers maximize applications performance on the Solaris OS running on the Intel Xeon processor 5500 series. Intel and Sun believe that the combination of Sun's open source operating system and Java™ environments, along with Sun Studio and NetBeans™ development tools, running on Intel Xeon processor 5500 series provide a leading edge platform for ISVs to develop and deliver applications. Sun Studio development tools provide C/C++/Fortran development environments for multicore environments, with specific improvements for performance, parallelism, and productivity.

Utilizing New Processor Instructions with the Solaris OS

Modern processors provide instruction set extensions like the Streaming SIMD Extensions (SSE) available on the Intel Xeon processor 5500 series. 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 non-temporal 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.

POPCNT counts the number of 1s in an operand.

Users on 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.

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

Useful documentation

Man pages for `dis(1)`, `isainfo(1)`, `file(1)`, `elfdump(1)`, `elfedit(1)`, `pargs(1)`, `cpuid(7D)`

Linker and Libraries Guide, Solaris 10 Software Developer Collection
http://docs.sun.com/app/docs/doc/817-1984

Linker and Libraries Guide, Solaris Express Software Developer Collection
http://docs.sun.com/app/docs/doc/819-0690

Intel 64 and IA-32 Architectures Software Developer's Manuals
http://www.intel.com/products/processor/manuals/

Improved Microvectorization and Automatic Parallelization

Intel Xeon processor 5500 series provide 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 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, Sun Studio users need to specify `-xvector=simd` in their compiler options.

In Sun Studio, microvectorization is applied on loops whenever safe. It can pack several iterations into one iteration, depending on 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 5500 series features Intel Hyper-Threading Technology, which provides two hardware threads per core—eight hardware threads in a four-core processor.

To speed up serial application performance on such multi-threaded chips, 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, Sun Studio users need to specify `-xautopar` in their compiler options. To enable reduction-style parallelization, 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 5500 series, microvectorization is improved continually. Here are some new improvements in Sun 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.
- Table-driven type selection and code generation for easy maintenance and future extensibility.
- Improved loop transformation to allow 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.

Automatic parallelization is also improved continuously from release to release. Here are some new improvements in Sun Studio.

- Array reduction is extended to cover more cases.
- More improvement in general data dependence analysis which leads more loops potentially being parallelized.
- Parallelization of loops with wrap-around data dependencies — more loops can be parallelized.
- More code transformation improvement like code specialization and loop fusion to enable more profitable loop parallelization.

Useful documentation

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture, Chapter 10 – 12. http://www.intel.com/products/processor/manuals/
Sun Studio 12 Collection. http://docs.sun.com/app/docs/coll/771.8?l=en

Instruction Selection

The new Intel Xeon processor 5500 series have 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.

```
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 5500 series 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, there are other issues that 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 5500 series 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.

```
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 an %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% 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 an %xmm register without causing a stall. Also, feeding some store units 64 bits at a time works better.

The Intel Xeon processor 5500 series deserve special mention because Intel has improved the microcode to deal with many of these issues. In the case of the Intel Xeon processor 5500 series, 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: 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 5500 series will execute the Core2/Penryn load/store combinations quite well. If there are Pentium®4, Core2 and Intel Xeon processor 5500 series in 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.

	-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 compiler also must be careful with generating code for arithmetic conversion operations on the new Intel Xeon processor 5500 series. 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.

The table below shows the 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 mem,xmm cvtdq2pd xmm,xmm	movdl mem,xmm cvtdp2pd xmm,xmm	xorps xmm,xmm cvtsi2sd mem,xmm	cvtsi2sd mem,xmm
	reg	movdl gpr,xmm cvtdq2pd xmm,xmm	movdl gpr,xmm cvtdq2pd xmm,xmm	xorps xmm,xmm cvtsi2sd gpr,xmm	cvtsi2sd gpr,xmm
long to double	mem	cvtsi2sdq mem,xmm	xorps xmm,xmm cvtsi2sdq mem,xmm	xorps xmm,xmm cvtsi2sdq mem,xmm	cvtsi2sdq mem,xmm
	reg	cvtsi2sdq gpr,xmm	xorps xmm,xmm cvtsi2sdq gpr,xmm	xorps xmm,xmm cvtsi2sdq gpr,xmm	cvtsi2sdq gpr,xmm
int to float	mem	movdl mem,xmm cvtdq2ps xmm,xmm	movdl mem,xmm cvtdq2ps xmm,xmm	movdl mem,xmm cvtdq2ps xmm,xmm	movdl mem,xmm cvtdq2ps xmm,xmm
	reg	movdl gpr,xmm cvtdq2ps xmm,xmm	movdl gpr,xmm cvtdq2ps xmm,xmm	movdl gpr,xmm cvtdq2ps xmm,xmm	movdl gpr,xmm cvtdq2ps xmm,xmm
long to float	mem	xorps xmm,xmm cvtsi2ssq mem,xmm	xorps xmm,xmm cvtsi2ssq mem,xmm	xorps xmm,xmm cvtsi2ssq mem,xmm	xorps xmm,xmm cvtsi2ssq mem,xmm
	reg	xorps xmm,xmm cvtsi2ssq gpr,xmm	xorps xmm,xmm cvtsi2ssq gpr,xmm	xorps xmm,xmm cvtsi2ssq gpr,xmm	xorps xmm,xmm cvtsi2ssq gpr,xmm
double to float	mem	movsd mem,xmm cvtpd2ps xmm,xmm	movsd mem,xmm cvtpd2ps xmm,xmm	movsd mem,xmm cvtpd2ps xmm,xmm	xorps xmm,xmm cvtsd2ss mem,xmm
	reg	cvtsd2ss xmm,xmm	cvtsd2ss xmm,xmm	cvtsd2ss xmm,xmm	xorps xmm,xmm cvtsd2ss xmm,xmm
float to double	mem	movss mem,xmm cvtps2pd xmm,xmm	movss mem,xmm cvtps2pd xmm,xmm	movss mem,xmm cvtps2pd xmm,xmm	cvtss2sd mem,xmm
	reg	cvtss2sd xmm,xmm	xorps xmm,xmm cvtss2sd xmm,xmm	xorps xmm,xmm cvtss2sd xmm,xmm	cvtss2sd xmm,xmm

Note: 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 5500 series 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 Sun Studio

Introduction

The Intel Xeon Processor 5500 Series features Hyper-Threading Technology, which features two hardware threads per core—a four core processor provides eight hardware threads per processor.

In order to take advantage of the parallelism offered by these multi-threaded architectures and boost performance, one needs to write multi-threaded applications. In a multi-threaded application multiple threads work together on executing the code in the program, thus speeding up the execution. One way to write multi-threaded applications is by using OpenMP.

What is OpenMP?

OpenMP has emerged as the de-facto standard for writing multi-threaded applications. OpenMP is an Application Programming Interface (API) that can be used to explicitly specify multi-threaded shared-memory parallelism in C, C++, and Fortran programs.

The OpenMP API is composed of three components:

- Compiler directives
- Runtime library routines
- Environment variables

The main motivations for using OpenMP are performance, scalability, portability, and standardization. With a relatively small amount of coding effort, a programmer can write multi-threaded applications to run on multi-threaded machines. Starting with a sequential program, the programmer can incrementally insert directives in the code to parallelize the program.

OpenMP has a rich set of directives that the programmer can use to specify parallelism in a 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 unique sentinels `(!$omp, c$omp, or *$omp)`.

The essential directive for creating threads is the `PARALLEL` directive. The `PARALLEL` directive defines a region of code to be executed in parallel by multiple threads. All the threads participating in the execution of the parallel region will execute the same region of code. In effect, the region of code is replicated across the threads.

Since OpenMP is based on the shared-memory programming model, variables in a parallel region are shared by default. OpenMP data scope attribute clauses can be used to explicitly define the scope of variables. These clauses include the `SHARED`, `PRIVATE`, `FIRSTPRIVATE`, `LASTPRIVATE`, and `REDUCTION` clauses.

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

OpenMP specification version 3.0

The OpenMP specification is the definitive reference on OpenMP. 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 (released May 2008). The 3.0 specification includes the following new features:

- Tasking: The TASK directive facilitates parallelizing applications where units of work are generated dynamically (as in a recursive structure or a while loop).
- Loop collapse: The COLLAPSE clause instructs the compiler to collapse perfectly nested loops and then parallelize the resulting loop.
- Nested parallelism support: The 3.0 specification allows better control over nested parallel regions, and provides new API routines to determine nesting structure.

Sun Studio software

The Sun Studio software is a comprehensive, integrated set of compilers and tools that enable the development and deployment of applications on a variety of platforms (SPARC, Intel, and AMD chips; Solaris and Linux Operating Systems). Sun Studio software is available for free.

The Sun Studio compilers (C, C++, and Fortran) support the OpenMP specification version 3.0. To compile 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, the Sun Studio software includes a variety of tools that facilitate OpenMP programming. Some of these tools are described below.

OpenMP debugging

The Sun Studio dbx tool can be used to debug C, C++, and Fortran 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`, and so on, variables for a given thread.

Data race and deadlock detection

The Sun 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 for writing.
- The threads are not using any exclusive locks to control their accesses to that memory.

When these three conditions hold, the order of accesses is non-deterministic, and the computation may give different results from run to run depending on that 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.
- Two or more threads form a circular chain in which each thread waits for a lock which 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 Thread Analyzer can detect potential deadlocks that may not have actually occurred in a particular run, but may occur in other runs.

OpenMP performance analysis

The Collector and Performance Analyzer are a pair of tools in Sun 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, 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

OpenMP is the de-facto standard for writing multi-threaded applications to run on multi-threaded machines. 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.

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

Useful documentation

1. *Official OpenMP Architecture Review Board web page*
   http://www.openmp.org
2 *OpenMP specification version 3.0*
   http://www.openmp.org/mp-documents/spec30.pdf
3. *Sun Studio Software*
   http://www.sun.com/software/products/studio/index.html
4. *Sun Studio 12 Collection Manuals*
   http://docs.sun.com/app/docs/coll/771.8
5. *Sun Studio OpenMP Wiki*
   http://wikis.sun.com/display/openmp/Sun+Studio+OpenMP

Appendices

About the Authors

NUMA and Multithreading

Jonathan Chew is a Staff Engineer and the Technical Lead for Solaris CMT and NUMA at Sun Microsystems. He received a bachelors degree in Applied Math (Computer Science) at UC Berkeley.

Performance Counters on Solaris, and Utilizing New Processor Instructions with the Solaris OS

Kuriakose Kuruvilla is a Kernel Engineer at Sun Microsystems and works on enabling OpenSolaris support for Intel processors. He received an MSE from Johns Hopkins University and a BTech from University of Kerala.

Power Aware Dispatcher

Eric Saxe is a Staff Engineer and the Technical Lead for Solaris Advanced Power Management at Sun Microsystems. He received a Bachelor of Science Degree in Computer Engineering from the University of California, San Diego.

PowerTOP for OpenSolaris

Rafael Vanoni is a Kernel Engineer at Sun Microsystems and works on enabling OpenSolaris support for Intel processors. He received a Bachelor's of Science at Federal University of Rio Grande do Sul (UFRGS).

Fast Reboot, and Support for Microcode Updates on Intel Processors

Sherry Moore is a Senior Staff Engineer in the Solaris Core Kernel Group. She has been with Sun Microsystems since 1997, and led various software teams for both SPARC and x86. She is currently the technical lead for the Solaris Engineering's x86 Kernel Team.

Juanita Heieck is a Senior Technical Writer in the Sun Learning Services organization at Sun Microsystems. She writes basic and advanced system administration documentation for a wide range of Solaris features, including booting, networking topics, and printing.

Improved Microvectorization and Automatic Parallelization

Yi Gao is a Software Engineer in C, C++, Optimization and Debugging team. He joined Sun in 2005 and since that time, he has been working on compiler optimization, microvectorization and performance tuning.

Yonghong Song received a Ph.D. degree in computer science from Purdue University. He joined Sun Microsystems in 2001. His work focuses on compiler optimizations, specifically in the area of automatic parallelization, memory and cache locality enhancement, data prefetching and interprocedural analysis and optimization.

Instruction Selection

David Seberger is the Technical Lead on SunStudio x86. David has authored and coauthored several patents relating to multiprocessing technology. David received a Master's degree in Computer Science from University of California, Davis, and undergraduate degrees in mathematics and computer science from U. C. Irvine.

OpenMP 3.0 Development with Sun Studio

Nawal Copty leads the OpenMP project at Sun. She is Sun's representative on the OpenMP Architecture Review Board (ARB) and the OpenMP language committee. She currently holds the post of Secretary of the ARB. Nawal received a Ph.D. degree in Computer Science from Syracuse University in 1995. Her interests include compilers, parallel algorithms, and languages and tools for multi-threaded programming.

Acknowledgements

Maxim Alt, Intel Corporation
Robert Kasten, Intel Corporation
Greg O'Keefe, Intel Corporation
Frank Wang, Intel Corporation
Chris Baker, Sun Microsystems
Scott Crase, Sun Microsystems
Ikroop Dhillon, Sun Microsystems
Adrian Frost, Sun Microsystems
Darrin Johnson, Sun Microsystems

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The docs.sun.com Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is *http://docs.sun.com*

To reference Sun BluePrints Online articles, visit the Sun BluePrints Online Web site at: *http://www.sun.com/blueprints/online.html*

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 1-650-960-1300 or 1-800-555-9SUN (9786) Web sun.com

Part Number: 820-7707-10 Printed in USA 03/09