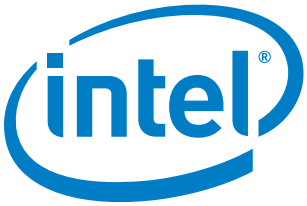
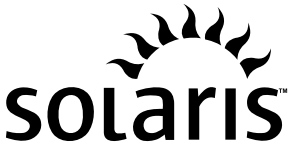


White Paper

Telecom and Data Center Application Migration Sun* Solaris* Operating Systems Intel® Multi-core Processors



A Migration Guide for Porting Applications on the SUN* Solaris* OS from SPARC* to Intel® Architecture Platforms

Disclaimer and Legal Information

Introduction

This migration guide discusses the process for porting C/C++ applications running on the Sun* Solaris* operating system (OS) from SPARC* platforms to Intel® architecture platforms. It outlines a number of possible software portation issues; however, a typical migration effort is unlikely to encounter all of them. Although this guide is directed towards software portation, some of the issues presented in this white paper are also relevant to other aspects of SPARC to Intel architecture platform migration.

Many equipment manufacturers are porting their telecom and data center applications running on Solaris to Intel® multi-core processor-based platforms to benefit from industry leading performance and greater performance per watt. Solaris delivers a consistent computing environment for business critical applications and infrastructure from departmental servers up to massive, clustered servers. The combination of the Solaris operating systems and Intel® processors enables unmatched performance, compact footprint, energy efficiency and simplified manageability.

This document provides engineers with useful migration planning information, a list of items worthy of consideration, code examples and URLs to on-line resources that can assist in the implementation. When text is underlined in the white paper, readers can find a corresponding URL link in the last section. It is assumed that any operating system version differences have already been addressed by the migration team.

Additional information is contained in another white paper describing an actual migration of a wireless infrastructure application by Synapse Mobile Networks* (the URL is listed at the end of this white paper), a leading supplier of mobile device and user management systems, They ported an Equipment Identification Register (EIR) application to an Intel architecture-based platform running the Sun Solaris operating system.

Whereas different porting projects often have unique challenges, the effort to create new applications for an Intel architecture-based platform is comparable for most popular operating systems, such as Sun Solaris, Linux* and Windows*.

Additionally, the migration of C/C++ applications, running on Intel architecture-based platforms, from Linux or Windows to Sun Solaris does not face the issues discussed in this document.

Why Migrate

Intel architecture has established itself as a proven leader in performance and innovation over its entire history. Intel continues to offer a strong roadmap with products that are optimized for performance, power and value. Intel's involvement in developing and driving standards-based platforms has dramatically altered the computer industry. As a result, many equipment manufacturers are deploying standards-based open architecture, which offers compatibility with legacy applications.

Many equipment manufacturers find software development has a significant impact on meeting their time to market objectives. To help speed the software development process, Intel provides a full line of development tools for implementing, debugging, and tuning software that optimize performance and ensure code correctness. These tools, along with open source initiatives, such as Open Solaris and strong support for development tools from major independent software vendors (ISVs), help developers reduce their effort and save time. Additionally, Intel initiatives to drive a strong ecosystem of independent hardware vendors (IHVs) and various form factor standards make it easier for developers to focus on their unique solution and enhance their intellectual property.

General Porting Issues

When porting an application to a new hardware platform, engineers must resolve issues across several development areas, including the following which are addressed in this document.

Installation Issues

- Root partition in a standard installation is often too small for development and should be increased (Suggestion: greater than 20gigabytes).
- Sun Studio should not be installed by default.

Developers are likely to need the Software Companion, which is located in the same location as the download for the operating system.

Device Issues

Device naming may have changed, which implies hardware dependent scripts have to be changed to accommodate the new device names.

Build Environment Issues

Before actually porting any source code, the first step is to locate the correct versions of all third-party utilities and libraries needed to build the application. Common examples are:

- source control system (e.g., Subversion, Perforce, ClearCase, and CVS)
- developer tools (e.g., Sun Studio, emacs, vim, xemacs and purify)
- build utilities (e.g., Sun Studio, Tcl/Tk, flex, bison and gmake)
- licensing, graphics, or other third-party libraries (e.g., zlib, qt)

Many of the open source utilities used on other platforms are already integrated in Solaris in /bin or available in /sfw. Many others are available in source or executable form. See Appendix A for a list of some of these utilities.

An application may also depend upon the availability of a third party component that needs to run on the same platform. Please verify key Solaris components are available for Intel architecture using the list of third party Solaris applications identified in Solaris Ready Applications and Solutions to check for availability.

Executable Locations

For Solaris, $PATH on both Intel architecture and SPARC systems should include, among others:

- /usr/bin
- /usr/ucb
- /usr/openwin/bin
- /usr/ccs/bin
- /usr/sfw/bin
- /opt/*/bin

Library Locations

The runtime libraries are found in the corresponding /lib directories (as well as /lib).

For historical reasons, Solaris also provides include files and runtime libraries compatible with SunOS* 4.X, in */usr/ucbinclude* and */usr/ucblib*. For new ports to Solaris, these ucb functions should be avoided in preference to the normal system routines.

Other small interface differences will be obvious during compiling and linking, which makes them comparatively easy to find and fix. Some examples are:

- 64-bit Solaris does not provide static versions of the 64-bit system libraries.
- The global variable *sys_errlist* is not visible in 64-bit Solaris, so *strerror* must be used instead.

Run-time Differences

Some software issues may not be obvious until run-time. While these issues may not be due to platform differences (e.g., Intel architecture versus SPARC), they may be caused by latent bugs in the application that are uncovered during the porting process. For example, a bad pointer or buffer over-run corrupting the heap may become evident after porting. These problems may require manual investigation using a debugger; however, Solaris also provides libumem, which is an alternate heap implementation providing the option of run-time consistency checks. For a more complete description and an example of using it to uncover a heap problem, please read Identifying Memory Management Bugs Within Applications using the libumem Library.

Additional concerns

The web site Freeware List for Intel and Solaris 10 contains many additional free packages for Solaris 10 on Intel architecture. The website has extensively documented the dependencies for building and installing these packages. Developers will also need to set their environmental variable PKG_CONFIG_PATH to include the directory where the new packages (sometimes just newer versions) are installed (by default /usr/local). This allows the configure script with the package to correctly identify which versions of which packages are available.

Since the default path for the package installation is /usr/local, developers must modify the environmental variable LD_LIBRARY_PATH to specify /usr/local/lib first. This causes the program loader to load the newly installed libraries, rather than possibly installed older versions, causing program failure. Also, some of the packages (e.g., wireshark) require a program (sed) from the gnu derivation, rather than the standard Solaris version. This necessitates /usr/xpg4/bin to be included on the environmental variable PATH before /usr/bin and /usr/ucb.

Compiler Related Issues

Choosing a compiler

Before considering the details of compiler differences, developers must first determine which compiler(s) to use. For C, developers can mix object code from compilers from different vendors. However, C++ compilers do not implement identical ABI's (Application Binary Interface), so the entire application should be built with the same brand of C++ compiler. For example, if the application depends on a library built with GNU* C++, it should be built with a GNU C++ compiler as well.

If none of these dependencies are applicable, the obvious tools choice for a Solaris port would be the Sun Studio compilers, debuggers and analyzers. Consider that for C++, changing to a new compiler can sometimes be as difficult as porting to a new OS.

Compiler options

When migrating to the Sun Studio compilers, the first visible difference is they accept different command line options. This is a minor technical hurdle, but it may require some parameterization of the existing Makefiles. Refer to the Compiler Option appendix in the C User's Guide and in the C++ User's Guide for an explanation of the compiler switches in the Sun Studio compilers.

After the port is functional, please reference the following paper to find the right options for your production build: Selecting The Best Compiler Options.

Engineers using Sun Studio compilers for both platforms, Intel architecture and SPARC, should be be prepared for some differences between them. For instance, using –cg89 with C++ on SPARC is the same as –xcg89 in cc. It is used for improving the runtime performance for SPARC platforms. On an Intel architecture platform, it generates a warning.

To avoid the warning, this compiler option should not be used on an Intel architecture platform. See Appendix B of this document for a more extensive list of compiler options supported on SPARC architecture that are not supported on Intel architecture. Also, there are options available on Intel architecture which are not supported on SPARC. For instance -386 is used for 80386 based instruction set and optimizations. See Appendix C of this document for a more extensive list. Furthermore, some options are supported by both architectures, but are interpreted differently. For instance, specifying –fnonstd expands to –fns –ftrap=common on SPARC, while it expands to –ftrap=common on Intel architecture. See Appendix D of this document for a more extensive list.

Conditional Compilation

If the application has been previously ported to some alternate operating system, hardware platform or compiler, then it probably already contains #ifdef's to isolate the port-specific code sequences. This makes porting to Solaris easier for a couple of reasons. First, code which runs on multiple platforms has already been generalized away from platform-specific dependencies. Second, the specific areas which have been previously #ifdef'd identify areas that warrant additional attention and inspection during the porting process.

For the common case where an application has already been released on Solaris/SPARC, porting to Solaris/Intel architecture may mostly be an exercise in selecting the correct set of #ifdef's from the existing sequences.

```
#ifdef __sparc
... big-endian sequence
#elif defined(__i386) || defined(__x86_64__)
... little-endian sequence
#endif
```

Language Issues

Current C++ compilers recognize slightly different definitions for the C++ language. They all strive to conform to the ISO C++ standard, but most compilers support some unique extensions while the ISO standard allows some features to be implementation defined. If such features are used in the source code, either purposefully or inadvertently, they must be dealt with during porting process.

Later releases of Sun Studio incorporate many of the GNU extensions, so using a newer version of Sun Studio will reduce the number of porting issues. However, some issues must be found and fixed manually.

Most of these language differences will cause the compiler's diagnostics utility to issue an error/warning message that describes the problem. The code can then be rewritten to match the well-defined areas of the ISO Standard. Sometimes the problem is sufficiently subtle that it requires research to understand what is wrong or how to fix it. For those so inclined, a copy of the C++ standard is the definitive source for investigating the language rules. However, an easier and often effective technique is to use an internet search engine to find discussions about the error message. Finally, if developers are unsure whether the compiler is correctly accepting or rejecting a particular language construct, they can submit questions to the Sun Developer Network Tools Forum.

Library Issues

When porting C++ using Sun Studio, developers must decide whether to use the default C++ Standard Library or the newer stlport4 library. The newer library provides better standards conformance and often better performance, but may require source changes (thus adding to the porting effort). For a discussion of the nuances of using and packaging the STLport library, see Using and Redistributing Sun Studio Libraries in an Application.

Hardware Architecture

Instruction Set Architecture

Since Solaris and Intel architecture platforms employ different instruction set architectures, any hand-generated assembly code must be converted. While there's no shortcut for removing all assembly code, there are several important cases where it may make sense to replace, rather than port, the assembly code.

- If the code was originally written in assembly for performance reasons, hardware and compiler improvements may now permit it to be rewritten in C or C++.
- If the assembly code is used to traverse the execution stack, it may now be rewritten on Solaris using the walkcontext function. In some situations, it may even be sufficient to exec the pstack utility.
- For cases of fine-grained parallelism, which may once have required hand-written spin locks, it may be possible to use the current implementation of pthread_mutex_lock. If spin locks are absolutely needed, another solution would be to use pthread_spin_lock.
- SPARC atomic must be replaced (Atomic SPARC: Using the SPARC Atomic Instructions). Arithmetic and logical atomic sequences may be replaced by calls to the atomic_ops.

One other issue comes from the functional differences in the graphics and vector capabilities of the underlying architecture (for example, Streaming SIMD Extensions (SSE) on Intel architecture or VIS on SPARC). Generally, these differences should be hidden within libraries (for example, Sun Performance Library).

Endianness

Endianness refers to the ordering of individual bytes within multi-byte data. This can be a porting issue when data is reinterpreted as a different type within a single process (for example, if an array of char is interpreted as type long) or between multiple processes when the memory image of a multi-byte datatype is written directly to a file, shared memory or a socket.

SPARC is big endian (where the most significant byte of an integer is stored in the lowest address), while Intel architecture is little endian (where the least significant byte of an integer is stored in the lowest addressed location). This issue is usually resolved by changing the application to always use a consistent byte ordering for shared multi-byte data.

Some porting issues due to endianness are:

- Data being sent over the network, if NBO (network byte order) is not used. In this case, all data being sent should be converted to use NBO. The minor modifications are bold and in italics.

  All four examples (server-bad.c, client-bad.c, server.c and client.c use test-Network-Code.h) which are at test-Network-Code.h.

  If a SPARC-based system (big endian) sends using (full code is at server-bad.c)

  Open and bind a socket. Then listen and accept messages. Use this information to send data back to client.

```
myData. aLong = 1; /* bad code */
strcpy(&myData.charArray[0], “123”);
```

  SPARC memory has

  myData.aLong (hex): 00 00 00 01
  myData.charArray(hex): 31 32 33 00

  Now send this data to the client.

  If an Intel architecture-based system (little endian) receives using (full code is at client-bad.c). Open a socket, determine the server and connect to it. Now send a message to the server, who will use the information to send the data back to the client. Next read from the socket and copy the data out.

```
strncpy((char *)&myData, (const char *) buf,
sizeof(myData));
```

  Intel architecture (little endian) memory is
  aLong(hex): 00 00 00 01
  *charPtr(hex): 31 32 33 00

  Which interprets aLong as 2^24

```
printf(“aLong=%08lx\n”, myData.aLong);
```

  Corrected server code (server.c) is:

  Open and bind a socket. Then listen and accept messages. Use this information to send data back to client. Use the function htonl (Host TO Network Long) to guarantee NBO of the data. Please note, on a big endian system (SPARC) this is a no-op, which has no effect on performance. However, it also works on a little endian (Intel architecture) system.

```
myData.aLong = htonl((long)1);
strcpy(&myData.charArray[0], “123”);
```

  SPARC memory has

  myData.aLong (hex): 00 00 00 01
  myData.charArray(hex): 31 32 33 00

  Now send this data to the client.

Corrected Client code is (client.c)

First, there must be allocation for space to correct to little endian (Intel architecture), while nothing is needed for big endian (SPARC).

```
#ifdef __SPARC
    /* Nothing needed */
#else
#ifdef __x86_64
    long myALong;
#else
#ifdef __i386
    long myALong;
#endif
#endif
#endif
```

Open a socket, determine the server and connect to it. Now send a message to the server, who will use the information to send the data back to the client. Next read from the socket and copy the data out.

```
    strncpy((char *)&myData, (const char *) buf,
    sizeof(myData));
    printf("myData.aLong = %08lx\n", myData.aLong);
#ifdef __SPARC
    printf("aLong=%ld\n", myData.aLong);
#else
#ifdef __x86_64
    myALong = ntohl(myData.aLong);
    printf("aLong=%08lx\n", myALong);
#else
#ifdef __i386
    myALong = ntohl(myData.aLong);
    printf("aLong=%08lx\n", myALong);
#endif
#endif
#endif
```

- Data stored and read by different architectures. Raw data stored in a file is not transferable between Intel architecture and SPARC platforms.

  If a SPARC-based system (big endian) writes raw data to a file.

```
long aLong = 1;
char * charPtr = "123";
    SPARC memory has
            aLong (hex):   00 00 00 01
            *charPtr(hex): 31 32 33 00
FILE * filePtr=fopen("fubar", "r");
fwrite((const void *) &aLong, (size_)
    sizof(aLong), (size_t) 1, filePtr););
fwrite((const void *) charPtr, (size_) 1, (size_t)
    strlen(charPtr), filePtr););
fclose(filePtr);
```

If the file is read on Intel architecture (little endian) and no byte swapping is done.

```
long aLong ;
char charArray[4];
int index;
FILE * filePtr=fopen("fubar", "w");
fread((void *) &aLong, (size_) sizof(aLong),
    (size_t) 1, filePtr););
for (index = 0; index < 4; index ++)
{
    fread((void *) &charArray[index], (size_) 1,
    (size_t) 1, filePtr););
}
    Intel architecture memory has
            aLong(hex):    00 00 00 01
            charaRRAY(hex):        31 32 33 00
    Which interprets aLong as 2^24
fclose(filePtr);
```

Corrected code using NBO (big endian), which requires no changes to SPARC based code:

Writer use:

```
long aLong = 1;
char * charPtr = "123";
#ifdef __x86_64
uint32_t fixedALong = htonl((uint32_t) aLong);
#else
#ifdef __i386
uint32_t fixedALong = htonl((uint32_t) aLong);
#endif
#endif
FILE * filePtr=fopen("fubar", "w");
#ifdef __x86_64
fwrite((const void *) &fixedALong, (size_)
    sizof(aLong), (size_t) 1, filePtr););
#else
#ifdef __i386
fwrite((const void *) &fixedALong, (size_)
    sizof(aLong), (size_t) 1, filePtr););
#else
fwrite((const void *) &aLong, (size_)
    sizof(aLong), (size_t) 1, filePtr););
#endif
#endif
fwrite((const void *) charPtr, (size_) 1,
    (size_t) strlen(charPtr), filePtr););
fclose(filePtr);
```

Reader uses:

```
   long aLong ;
   ifdef __x86_64
 uint32_t aLongIn;
 #else
 #ifdef __i386
 uint32_t aLongIn;
 #endif
 #endif
 char charArray[4];
 int index;
 FILE * filePtr=fopen("fubar", "r");
 #ifdef __i386
 fread((void *) &aLongIn, (size_) sizof(aLongIn),
 (size_t) 1, filePtr););
 aLong=ntohl((uint32_t) aLongIn);
 #else
 #ifdef __i386
 fread((void *) &aLongIn, (size_) sizof(aLongIn),
 (size_t) 1, filePtr););
 aLong=ntohl((uint32_t) aLongIn);
 #else
 fread((void *) &aLong, (size_) sizof(aLong),
 (size_t) 1, filePtr););
 #endif
 for (index = 0; index < 4; index ++)
 {
   fread((void *) &charArray[index], (size_) 1,
 (size_t) 1, filePtr););
 ]
 fclose(filePtr);
```

- Data storage order. The order of data stored differs between the systems. For example using the following structure:

```
struct timeStruct {
char hour;
char minute;
char second;
char fill; /* To fill the structure out to be 4 bytes */
};
.
.
struct timeStruct startTime;
struct timeStruct endTime;
.
.
/*
* For better performance, compare all time
  components in single compare.
*/
If ((*(long *) &endTime) < (*(long *)
  &startTime))
{
  /* Time must have wrapped around */
}
```

For code to portable, individual fields should be checked

Please see the *Intel White Paper on Endianness* for complete details on software considerations related to microprocessor endian architecture and guidelines for developing endian-neutral code.

Data Alignment

Another processor related issue is data alignment, where multi-byte data needs to be stored on some minimally aligned address. As a general rule, SPARC platforms require data alignment to equal data size: two byte types should be on a two byte boundary, four byte types should be on a four byte boundary. On other processors, like Intel architecture, the hardware will handle misaligned data, but there may be a performance penalty.

The compilers will automatically arrange for data to be appropriately aligned. This is accomplished by adding padding around the initial location of the stack and heap and within structures, before static data locations. However, alignment problems can be introduced via casting, for example, by taking the address of an arbitrary element of a char array and interpreting it as the address of an element of type double.

The highest performance and most general solution is to change the logic of the application to maintain correct alignment. However, on a SPARC platform it may be reasonable (at least during the initial stages of the port) to have the Sun Studio compilers convert multi-byte loads and stores into a sequence of smaller accesses via the -xmemalign option. For example, developers can use -xmemalign=2i to generate loads/stores no larger than two-byte (with the assumption of no more than two-byte alignment).

I/O Architecture

SPARC architecture treats PCI I/O space the same as an access to memory, while Intel architecture uses special IN and OUT instructions. All code accessing PCI I/O space must be ported to accommodate this difference.

Alternatives to Porting

There are also a few alternatives to porting. Developers can use technologies that allow applications compiled for one CPU and operating system to run on platforms using different CPUs and operating systems, such as those offered by *Transitive*. They have three versions of their QuickTransit product for running Solaris/SPARC applications on other platforms. These are QuickTransit Workstation, Server and Legacy. Does any part of the code consist of applications that are not performance critical? If yes, consider Transitive's QuickTransit for the easiest migration effort. Also, as mentioned earlier, *Solaris Containers* is another alternative.

Conclusion

In conclusion, there are a variety of potential problem areas when porting to Solaris applications from SPARC to Intel architecture platforms, which can be handled by proper planning. Although the migration task may seem daunting, it is usually not the case. It would be very rare for any portation to encounter all of the potential problems described in this white paper.

Many equipment manufacturers and IT departments are migrating business-critical applications, such as telecom and data center, to more cost-effective, highly available platforms. Their software and platform migration teams can benefit from the insights offered by this white paper and the referenced online sources. With proper planning, developers are better able to meet their project timelines and ensure their applications operate properly and perform well.

Migration Checklist

Developers planning a migration effort should consider the following points.

Product or Application Description

What are the basic capabilities of this product or application?

Does the software abstract the hardware and how is memory defined?

Current Versions

What is the version of Solaris being used?

What is the development or build environment (i.e. Sun Studio 10, GNU)?

If the build environment is Sun-specific, which version?

If GNU toolset is used to build, which versions of tools were used (i.e. gcc 3.4.3)?

What is the current hardware configuration being used?

Availability of Source Code

- Is the complete code for the application/driver/etc. available?
- Is the complete code available on a media in a tar or cpio format?
- Is the complete code available from a source code control system?

Assembly Code

If any of the code is in assembly, it will have to be re-written.

Build Environment

How is the product currently built?

What version control system is currently used (SVN, CVS)?

Have any in-house build tools been developed?

If yes, they need to go through the same migration.

Is information on completely building the application available?

Does the build use third-party libraries?

If yes, are they available on new environment?

Is there a document which specifies the list of items to be built on the target?

Appendix A. Links to tools

Compiler related:

Sun Studio Compilers and Tools

C User's Guide

C++ User's Guide

Selecting The Best Compiler Options

64-bit Intel Architecture Migration, Debugging, and Tuning, With the Sun Studio

Compiler Differences Between Solaris OS, SPARC Platform and Intel Architecture Platform

Tools:

Solaris Containers

Transitive

Solaris Operating System - Freeware

Solaris Freeware Project

Freeware List for Intel and Solaris 10

Blastwave.org - An OpenSolaris Community Site

Identifying Memory Management Bugs Within Applications using the libumem Library

walkcontext

pstack

pthread_mutex_lock

pthread_spin_lock

atomic_ops

PowerTOP

Solaris Ready Applications and Solutions

Appendix B. Compiler options existing on SPARC* not on Intel® architecture

Compiler Option	Explanation	Behavior on Intel® architecture Version
`-cg89` (C++)	This option is the same as -xcg89 in cc. Used for improving runtime performance for the SPARC architecture. It compiles the C++ code for generic SPARC architecture. Equivalent	Warning issued by C++ to -xtarget=ss2. compilers [1}
`-cg92` (C++)	This option is the same as -xcg92 in cc. Used for improving runtime performance for the SPARC architecture. It compiles the C++ code for SPARC V8 architecture. Equivalent to -xtarget=ss1000.	Warning issued by C++ compiler [1]
`-dalign` (C)	Equivalent to -xmemalign=8s. This flag allows the user to specify the maximum memory alignment of 8 bytes to be assumed by the compiler in indeterminable situations. It also specifies that SIGBUS be raised when misaligned memory access takes place.	Option ignored [3]
`-fns`	Turns on the SPARC nonstandard floating-point mode. When nonstandard mode is enabled, floating-point arithmetic may produce results that do not conform to the requirements of the IEEE 754 standard.	Option ignored [3]
`-misalign`	Equivalent to -xmemalign=1i. This flag allows the user to specify the maximum memory alignment of 1 byte to be assumed by the compiler in indeterminable situations. This option specifies that access be interpreted and execution be continued, when a misaligned memory access takes place.	Option ignored [3]
`-misalign2` (C)	Equivalent to -xmemalign=2i. This flag allows the user to specify the maximum memory alignment of 2 bytes to be assumed by the compiler in indeterminable situations. This option specifies that SIGBUS be raised when a misaligned access occurs.	Option ignored [3]
`-xalias_level`	Used to determine the assumptions that can be made by the compiler in order to perform optimizations (using the -x0 option) using type-based alias-analysis.	Warning issued by C compiler [3]
`-xautopar` (C)	This option turns on automatic parallelization for multiple processors used in the SPARC architecture. It also does dependence analysis (analyzes loops for inter-iteration data dependence) and loop restructuring.	Option ignored [3]
`-xcache` (C++)	This option defines cache properties for use by the optimizer.	Warning issued by C++ compiler [1]
`-xcg89` (C)	This option is used for improving runtime performance for the SPARC architecture. It compiles the C++ code for generic SPARC architecture. Equivalent to `-xtarget=ss2 (-xarch=v7 -xchip=old -xcache=64/32/1)`	Option ignored [3]
`-xcg92` (C)	This option is used for improving runtime performance for the SPARC architecture. It compiles the C++ code for SPARC V8 architecture. Equivalent to -xtarget=ss1000 `(-xarch=v8 -xchip=super -xcache=16/32/4:1024/32/1).`	Option ignored [3]
`-xcheck`	This option adds a runtime check for stack overflow.	Warning issued by C compiler [3]
`-xcode`	This option specifies the code address space.	Warning issued by C compiler[3]
`-xcrossfile` (C)	This option enables optimization and inlining across source files.	Option ignored [3]
`-xdepend` (C)	This option analyzes loops for inter-iteration data dependencies and does loop restructuring.	Option ignored [3]
`-xexplicitpar` (C)	This option generates paralleled code based on specification of #pragma MP directives.	Option ignored [3]
`-xhwcprof` (C)	This option enables compiler support for hardware counter-based profiling.	Option ignored [3]
`-xia` (C++)	This option links the appropriate interval arithmetic libraries and sets a suitable floating-point environment.	Warning issued by C++ compiler[1]
`-xipo`	This option performs inter-procedural optimizations.	Option ignored [3]
`-xjobs`	This option specifies the maximum number of components the compiler will fork in parallel.	Warning issued by C compiler[2]
`-xldscope`	This option is used to change the default linker scoping for the definition of extern symbols.	Warning issued by C compiler[2]
`-xlic_lib=sunperf`	This option links in the Sun-supplied performance libraries.	Option ignored [3]

Compiler Option	Explanation	Behavior on Intel® architecture Version
`-xlinkopt` (C)	This option instructs the compiler to perform link-time optimization on the resulting executable or dynamic library over and above any optimizations in the object files.	Warning issued by C compiler[2]
`-xloopinfo` (C)	This option shows which loops are paralleled and which are not.	Warning issued by C compiler[2]
`-xmemalign`	This option specifies maximum assumed memory alignment and behavior of misaligned data accesses.	Warning issued by C compiler[2]
`-xMerge`	This option merges data segments into text segments.	Option ignored [3]
`-xopenmp`	This option enables explicit parallelization with OpenMP* directives.	Warning issued by C compiler[2]
`-xpagesize`	This option sets the preferred page size for the stack and the heap.	Warning issued by C compiler[2]
`-xpagesize_heap`	This option sets the preferred page size for the heap.	Warning issued by C compiler[2]
`-xpagesize_stack`	This option sets the preferred page size for the stack.	Warning issued by C compiler[2]
`-xport64` (C++)	This option helps to debug code being ported to a 64-bit environment.	Warning issued by C++ compiler[1]
`-xparallel` (C)	This option parallelizes loops both automatically by the compiler and explicitly specified by the programmer.	Option ignored [3]
`-xprefetch`	This option enables prefetch instructions on those architectures that support prefetch.	Warning issued by C compiler[2]
`-xprefetch_level`	This option controls the number of pre-fetch instructions as determined with `-xprefetch=auto.`	Warning issued by C compiler[2]
`-xprofile_ircache`	This option is used with `-xprofile=collect,` or `-xprofile=use` to improve compilation time during the use phase by reusing compilation data saved from the collect phase.	Warning issued by C compiler[2]
`-xprofile_pathmap`	This option is used with -xprofile=use and it uses the prefix of the UNIX* pathname of a directory tree in which object files were compiled.	Warning issued by C compiler[2]
`-xreduction` (C)	This option turns on reduction recognition during automatic parallelization.	Option ignored [3]
`-xregs`	This option specifies the usage of registers for the generated code.	Warning issued by C compiler[2]
`-xrestrict` (C)	This option treats pointer-valued function parameters as restricted pointers.	Option ignored [3]
`-xsafe=mem`	This option allows the compiler to assume no memory-based traps occur.	Option ignored [3]
`-xspace`	This option passes the instruction to the compiler to not do optimizations or parallelization of loops that increase code size.	Option ignored [3]
`-xthreadvar`	This option controls the implementation of thread local variables.	Warning issued by C compiler[2]
`-xvector`	This option enables automatic generation of calls to the vector library functions.	Warning issued by C compiler[2]
`-xvis`	This option is used when the assembly-language templates defined in the VIS instruction-set Software Developers Kit (VSDK) are used.	Option ignored [3]
`-xvpara` (C)	This option warns about loops that have #pragma MP directives specified when the loop may not be properly specified for parallelization.	Option ignored [3]
`-Zll`(C)	This option creates the program database for lock_lint, but does not generate executable code.	Option ignored [3]

Key: (C) denotes compiler option used by cc (C compiler). (C++) denotes compiler option used by CC (C++ compiler). Otherwise the option is valid for both cc and CC (C and C++ compilers).

Notes:

[1] The C++ compiler option works only for the SPARC* platform and would give a compiler warning when used on the x 86 platforms. To avoid this warning, this compiler option should not be used on the x 86 platforms.

[2] The C compiler option works only for the SPARC platform and would give a compiler warning when used on the x 86 platforms. To avoid this warning, this compiler option should not be used on the x 86 platforms

[3] The option is valid only on the SPARC platform. While its use on the x 86 platforms does not generate any warning, this option is ignored during the generation of binaries on the x 86 platforms

Appendix C. Compiler options existing on Intel® architecture platforms and not on SPARC* platforms

Compiler Option	Explanation	Behavior on SPARC* Platform
`-386` (C++)	Same as -xtarget=386. Specifies the target platform as Intel386™ processor-based instruction set. and optimization	Warning issued by C++ compiler[1]
`-486` (C++)	Same as -xtarget=486. Specifies the target platform as Intel486™ processor-based instruction set and optimization.	Warning issued by C++ compiler[1]
`-fprecision`	This option initializes the rounding-precision mode bits in the floating-point control word to single (24 bits), double (53 bits), or extended (64 bits).	Option ignored[3]
`-fstore`	This option causes the compiler to convert the value of a floating-point expression or function to the type on the left side of an assignment rather than leave the value in a register when: The expression or function is assigned to a variable. The expression is cast to a shorter floating-point type.	Option ignored[3]
`-nofstore`	This option disables forced precision of an expression.	Option ignored[3]
`-x386` (C)	This option optimizes the code for the Intel® 80386 processor.	Option ignored[3]
`-x486` (C)	This option optimizes the code for the Intel® 80386 processor.	Option ignored[3]
`-xpentium`	This option optimizes the code for the Intel® Pentium® processor.	Warning issued by C compiler[2]

Key: (C) denotes compiler option used by cc (C compiler). (C++) denotes compiler option used by CC (C++ compiler). Otherwise the option is valid for both cc and CC (C and C++ compilers).

Notes:

[1] The C++ compiler option works only for the Intel® architecture platform and would give a compiler warning when used on the SPARC* platform. To avoid this warning, this compiler option should not be used on the SPARC platform.

[2] The C compiler option works only for the Intel architecture platform and would give a compiler warning when used on the SPARC platform. To avoid this warning, this compiler option should not be used on the SPARC platform.

[3] The option is valid only on the x 86 platforms. While its use on the SPARC platform does not generate any warning, this option is ignored during generation of binaries on the SPARC platform

Appendix D. Differing compiler options on both SPARC* and Intel® architecture platforms

Compiler Option	Explanation	Values for SPARC*	Values for Intel® architecture
`-Aname[(tokens)]` (C)	This option associates *name* with the specified *tokens* like a #define preprocessing directive.	The values for preassertions are: machine(sparc) cpu(sparc)	The values for preassertions are: machine(Intel386) cpu(Intel386)
`-Dname[=tokens]`	This option associates *name* with the specified tokens like a #define preprocessing directive.	The values are: sparc (not valid in-Xc mode) __sparc (valid in all modes)	The values are: Intel386 (not valid in-Xc mode) __i386 (valid in all modes)

`-fast`	This option selects a set of baseline options for optimizing benchmark applications.	The option expands to the following values on SPARC platform:
		`-fns`
		`-fsimple=2`
		`-fsingle`
		`-ftrap=%none`
		`-xalias_level=basic`
		`-xarch`
		`-xbuiltin=%all`
		`-xdepend`
		`-xlibmil`
		`-xmemalign=8s`
		`-xO5`
		`-xprefetch=`
		`auto,explicit`
			The option expands to the following values on Intel architecture platform:
			`-fns`
			`-fsimple=2`
			`-fsingle`
			`-ftrap=%none`
			`-nofstore`
			`-xarch`
			`-xbuiltin=%all`
			`-xdepend`
			`-xlibmil`
			`-xO5`

`-fnonstd`	This option causes nonstandard initialization of floating-point arithmetic hardware.	This option expands to:
		`fns`
		`-ftrap=common`
			This option expands to:
			`ftrap=common`

`-KPIC`	This option is used to compile source files when building a shared library.	Equivalent to: `-xcode=pic32`	Same as `-Kpic`
`-Kpic`	This option is used to compile source files when building a shared library.	Equivalent to: `-xcode=pic13`	Compiles with position-independent code.
`-PIC` (C++)	This option is used to compile source files when building a shared library.	Equivalent to: `-xcode=pic32`	Same as `-Kpic`
`-pic` (C++)	This option is used to compile source files when building a shared library.	Equivalent to: `-xcode=pic13`	Same as `-Kpic` Same as -Kpic

`-xarch`	This option specifies the instruction set architecture (ISA).	The values are:
		`generic`
		`generic64`
		`native`
		`native64`
		`v7`
		`v8a`
		`v8`
		`v8plus`
		`v8plusa`
		`v8plusb`
		`v9a`
		`v9b`
		`v9`
			The values are:
			`generic`
			`386`
			`pentium_pro`

Compiler Option	Explanation	Values for SPARC*	Values for Intel® architecture

`-xchip`	This option specifies the target processor for use by the optimizer.	The values are:
		`generic`
		`old`
		`super`
		`super2`
		`micro`
		`micro2`
		`hyper`
		`hyper2`
		`powerup`
		`ultra`
		`ultra2`
		`ultra2e`
		`ultra2i`
		`ultra3`
		`ultra3cu`
			The values are:
			`generic`
			`386`
			`486`
			`pentium`
			`pentium_pro`

`-xO`	This option optimizes the object code depending on the levels set.	The values are:
		`xO1: Does basic local optimization`
		`xO2: Does basic local and global optimization.`
		`xO3: Performs like -xO2, but also optimizes references or definitions for external variables.`
		`xO4: Performs like -xO3, but also automatically inlines functions contained in the same file for faster execution.`
		`xO5: Generates the highest level of optimization.`
			The values are:
			`xO1: Preloads arguments from memory and cross-jumping, as well as the single pass of the default optimization.`
			`XO2: Schedules both high- and low-level instructions and performs improved spill analysis, loop memory-reference elimination, register lifetime analysis, enhanced register allocation, and elimination of global common sub-expressions.`
			`xO3: Performs loop strength reduction, induction variable elimination, on top of the functions carried out by the -xO2 option.`
			`xO4: Performs loop unrolling, avoids creating stack frames when possible, and automatically inlines functions contained in the same file, on top of the functions carried out by -xO3.`
			`xO5: Generates the highest level of optimization.`

`-xtarget`	This option specifies the target system for instruction set and optimization.	The values are:
		`native`
		`native64`
		`generic`
		`generic64`
		`<<platform-name>>`
			The values are:
			`native`
			`generic`
			`386`
			`486`
			`pentium`
			`pentium_pro`

Key: (C) denotes compiler option used by cc (C compiler). (C++) denotes compiler option used by CC (C++ compiler). Otherwise the option is valid for both cc and CC (C and C++ compilers).

Appendix E. More Information

For more information on related topics, see the following.

Others:

Atomic SPARC: Using the SPARC Atomic Instructions

Intel White Paper on Endianness

OpenSolaris

Solaris ABI Program / Appcert

Sun Developer Network Tools Forum

Sun Performance Library

Using and Redistributing Sun Studio Libraries in an Application

Appendix F. Test Network Code

The appropriate code to be corrected is bold and in italics.

F.1. test-Network-Code.h

```
#define SERVER_PORT 54321
#define BUFFER_SIZE 1024

typedef struct
{
  long aLong ;
  char charArray[4];
} myDataStruc;
```

F.2. server-bad.c

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>

#include “test-Network-Code.h”

int
main(int argc, char * argv[], char ** envp)
{
  int sock;
  int msgsock;
  struct sockaddr_in server;
  struct sockaddr_in client;
  int clientLen;
  int rval;
  char buf[BUFFER_SIZE];
  myDataStruc myData;

    /* Open a socket, not bound yet.  Type is Internet TCP. */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      {
    perror(“Opening stream socket”);
    exit(1);
      }

  /*
    Prepare to bind.  Permit Internet connections from any client to our SERVER_PORT.
  */
  bzero((char *) &server, sizeof(server));
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_port = htons(SERVER_PORT);
  if (bind(sock, (struct sockaddr *) &server, sizeof(server)))
    {
      perror(«Binding stream socket»);
      exit(2);
   }

  /* Set the listen queue depth to 5, the maximum. */
  listen(sock, 5);
```

```
  /* Loop, waiting for client connections. */
  /* This is an interactive server. */
  while (1 == 1)
    {
      clientLen = sizeof(client);
      if ((msgsock = accept(sock, (struct sockaddr *) &client,
                    &clientLen)) == -1)
    {
      perror(«Accept»);
      exit(3);
    }
      else
    {
      if (clientLen != sizeof(client))
        {
          perror(«Accept overwrote sockaddr structure.»);
          exit(4);
        }

      do
        { /* Read from client until the connections closed */
          /* Prepare read buffer and read. */
          bzero(buf, sizeof(buf));
          if ((rval = read(msgsock, buf, BUFFER_SIZE)) < 0)
        {
          perror(«Reading stream message»);
          exit(5);
        }

          if (rval != 0)
        {/* Write back to client. */
               /* Setup the data to send */
                    myData. aLong = 1;
                    strcpy(&myData.charArray[0], “123”);
                SPARC memory has
                myData.aLong (hex):               00 00 00 01
                myData.charArray(hex):            31 32 33 00
                   printf(“aLong=%08lx\n”, myData.aLong);
            if (write(msgsock, (const void *) &myData, sizeof(myData))
                < 0)
              {
                perror(“Writing on stream socket”);
                exit(6);
              }
        }

        } while (rval != 0);
    }   /* else */
      close(msgsock);
    }

  exit(0);

}
```

F.3. client-bad.c

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <strings.h>
#include <netdb.h>
#include <unistd.h>

#include “test-Network-Code.h”

int
main(int argc, char * argv[], char ** envp)
{
  int sock;
  struct sockaddr_in server;
  struct sockaddr_in client;
  int clientLen;
  struct hostent *hostnamePtr;
  char buf[BUFFER_SIZE];
  myDataStruc myData;
  int i;   /* loop counter */

  if (argc != 2)
    {
      perror(“Usage: client hostname”);
      exit(1);
    }

  /* Open socket --- Internet TCP type. */
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
      perror(“Opening stream socket”);
      exit(2);
    }

  /* Prepare to connect to server. */
  bzero((char *) &server, sizeof(server));
  server.sin_family = AF_INET;
  if ((hostnamePtr = gethostbyname(argv[1])) == NULL) {
    sprintf(buf, “%s: unknown host\n”, argv[1]);
    perror(buf);
    exit(3);
  }
  bcopy(hostnamePtr->h_addr, &server.sin_addr, hostnamePtr->h_length);
  server.sin_port = htons((u_short) SERVER_PORT);

  /* Try to connect */
 if (connect(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
    {
      perror(«Connecting stream socket»);
      exit(4);
    }

  /* Determine what port client’s using. */
  clientLen = sizeof(client);
  if (getsockname(sock, (struct sockaddr *) &client, &clientLen))
    {
```

```
      perror(«Getting socket name»);
      exit(5);
    }

  if (clientLen != sizeof(client))
    {
      perror(«getsockname() overwrote name structure»);
      exit(6);
    }

  printf("Client socket has port %hu\n", ntohs(client.sin_port));

  /* Write out message. */
  if (write(sock, (const void *) &myData, sizeof(myData)) < 0)
    {
      perror(«Writing on stream socket»);
      exit(7);
    }
  /* Prepare our buffer for a read and then read. */
  bzero(buf, sizeof(buf));
  if (read(sock, buf, BUFFER_SIZE) < 0)
    {
      perror(«Reading stream message»);
      exit(8);
    }

  strncpy((char *)&myData, (const char *) buf, sizeof(myData));
         Intel architecture (little endian) memory is
                  aLong(hex):      00 00 00 01
                  *charPtr(hex):   31 32 33 00
         Which interprets aLong as 2^24
  printf("aLong=%08lx\n", myData.aLong);

  /* Close this connection. */
  close(sock);

}
```

F. 4. server.c

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <netinet/in.h>
#include <inttypes.h>

#include “test-Network-Code.h”

int
main(int argc, char * argv[], char ** envp)
{
  int sock;
  int msgsock;
  struct sockaddr_in server;
  struct sockaddr_in client;
  int clientLen;
  int rval;
  char buf[BUFFER_SIZE];
  myDataStruc myData;

    /* Open a socket, not bound yet.  Type is Internet TCP. */
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      {
    perror(“Opening stream socket”);
    exit(1);
      }

  /*
    Prepare to bind.  Permit Internet connections from any client
    to our SERVER_PORT.
  */
  bzero((char *) &server, sizeof(server));
 server.sin_family = AF_INET;
  server.sin_addr.s_addr = INADDR_ANY;
  server.sin_port = htons(SERVER_PORT);
  if (bind(sock, (struct sockaddr *) &server, sizeof(server)))
    {
      perror(«Binding stream socket»);
      exit(2);
    }

  /* Set the listen queue depth to 5, the maximum. */
  listen(sock, 5);

  /* Loop, waiting for client connections. */
  /* This is an interactive server. */
  while (1 == 1)
    {
      clientLen = sizeof(client);
      if ((msgsock = accept(sock, (struct sockaddr *) &client,
                    &clientLen)) == -1)
    {
      perror(«Accept»);
      exit(3);
```

```
    }
      else
    {
      if (clientLen != sizeof(client))
        {
          perror(«Accept overwrote sockaddr structure.»);
          exit(4);
        }

      do
        { /* Read from client until the connections closed */
          /* Prepare read buffer and read. */
          bzero(buf, sizeof(buf));
          if ((rval = read(msgsock, buf, BUFFER_SIZE)) < 0)
        {
          perror(«Reading stream message»);
          exit(5);
        }

          if (rval != 0)
        {/* Write back to client. */
               /* Setup the data to send */

                   myData.aLong = htonl((long)1);
                   strcpy(&myData.charArray[0], “123”);
                   printf(“aLong=%08lx\n”, myData.aLong);
            if (write(msgsock, (const void *) &myData, sizeof(myData)) < 0)
              {
                perror(“Writing on stream socket”);
                exit(6);
              }
        }

        } while (rval != 0);
    }   /* else */
       close(msgsock);
    }
  exit(0);
}
```

F. 5. client.c

```
#include<stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <strings.h>
#include <netdb.h>
#include <unistd.h>
#include <netinet/in.h>
#include <inttypes.h>

#include “test-Network-Code.h”

int
main(int argc, char * argv[], char ** envp)
{
  int sock;
  struct sockaddr_in server;
  struct sockaddr_in client;
  int clientLen;
  struct hostent *hostnamePtr;
  char buf[BUFFER_SIZE];
  myDataStruc myData;
  int i;   /* loop counter */
#ifdef __SPARC
  /* Nothing needed */
#else
#ifdef __x86_64
    long myALong;
#else
#ifdef __i386
    long myALong;
#endif
#endif
#endif

  if (argc != 2)
    {
      perror(“Usage: client hostname”);
      exit(1);
    }

  /* Open socket --- Internet TCP type. */
  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
      perror(“Opening stream socket”);
      exit(2);
    }

  /* Prepare to connect to server. */
  bzero((char *) &server, sizeof(server));
  server.sin_family = AF_INET;
  if ((hostnamePtr = gethostbyname(argv[1])) == NULL) {
    sprintf(buf, “%s: unknown host\n”, argv[1]);
    perror(buf);
    exit(3);
  }
  bcopy(hostnamePtr->h_addr, &server.sin_addr, hostnamePtr->h_length);
  server.sin_port = htons((u_short) SERVER_PORT);
```

```
  /* Try to connect */
  if (connect(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
    {
      perror("Connecting stream socket");
      exit(4);
    }

  /* Determine what port client's using. */
  clientLen = sizeof(client);
  if (getsockname(sock, (struct sockaddr *) &client, &clientLen))
    {
      perror("Getting socket name");
      exit(5);
    }

  if (clientLen != sizeof(client))
    {
      perror("getsockname() overwrote name structure");
      exit(6);
    }

  printf("Client socket has port %hu\n", ntohs(client.sin_port));

  /* Write out message. */
  if (write(sock, (const void *) &myData, sizeof(myData)) < 0)
    {
      perror("Writing on stream socket");
      exit(7);
    }
  /* Prepare our buffer for a read and then read. */
  bzero(buf, sizeof(buf));
  if (read(sock, buf, BUFFER_SIZE) < 0)
   {
      perror("Reading stream message");
      exit(8);
    }

  strncpy((char *)&myData, (const char *) buf, sizeof(myData));
  printf("myData.aLong = %08lx\n", myData.aLong);
#ifdef __SPARC
  printf("aLong=%ld\n", myData.aLong);
#else
#ifdef __x86_64
  myALong = ntohl(myData.aLong);
 printf("aLong=%08lx\n", myALong);
#else
#ifdef __i386
  myALong = ntohl(myData.aLong);
  printf("aLong=%08lx\n", myALong);
#endif
#endif
#endif

  /* Close this connection. */

  close(sock);

}
```

Appendix G. Addresses of links

This section provides the addresses of all links used in this document, in case this document is used in printed from, rather than online.

64-bit Intel architecture Migration, Debugging, and Tuning, With the Sun Studio
http://developers.sun.com/solaris/articles/amd64_migration.html

atomic_ops
http://docs.sun.com/app/docs/doc/819-2243/atomic-ops-3c?l=en&a=view&q=atomic_ops

Atomic SPARC: Using the SPARC Atomic Instructions
http://developers.sun.com/solaris/articles/atomic_sparc/

Blastwave.org - An OpenSolaris Community Site
http://www.blastwave.org/

C User's Guide
http://docs.sun.com/app/docs/doc/819-5265

C++ User's Guide
http://docs.sun.com/app/docs/doc/819-5267

Compiler Differences Between Solaris OS, SPARC Platform and Intel Architecture Platform
http://developers.sun.com/solaris/articles/x86_compiler_diffs.html

Freeware List for Intel and Solaris 10
http://www.sunfreeware.com/programlistintel10.html

Identifying Memory Management Bugs Within Applications using the libumem Library
http://access1.sun.com/techarticles/libumem.html

Intel White Paper on Endianness
http://www.intel.com/design/intarch/papers/endian.htm

OpenSolaris
http://opensolaris.org/os/

PowerTOP
http://www.lesswatts.org/projects/powertop/

pstack
http://docs.sun.com/app/docs/doc/817-5441/6mkt8ku11?l=en&a=view&q=pstack

pthread_mutex_lock
http://docs.sun.com/app/docs/doc/819-2243/pthread-mutex-lock-3c?l=en&a=view&q=mutex_lock

pthread_spin_lock
http://docs.sun.com/app/docs/doc/819-2243/6n4i099f1?l=en&a=view&q=pthread_spin_lock

Selecting The Best Compiler Options
http://docs.sun.com/source/820-5242/index.html

Solaris ABI Program / Appcert
http://www.sun.com/software/solaris/programs/abi/index.xml

Solaris Containers
http://www.sun.com/software/solaris/containers/index.jsp

Solaris Freeware Project
http://www.sunfreeware.com/

Solaris Operating System - Freeware
http://www.sun.com/software/solaris/freeware/index.xml

Sun Developer Network Tools Forum
http://forum.java.sun.com/index.jspa?tab=devtools

Sun Performance Library
http://docs.sun.com/source/806-3566/plug_title.html

Solaris Ready Applications and Solutions
http://www.sun.com/bigadmin/apps/data/views/all_applications_all_results.page1.html

Sun Studio Compilers and Tools
http://developers.sun.com/sunstudio/index.jsp

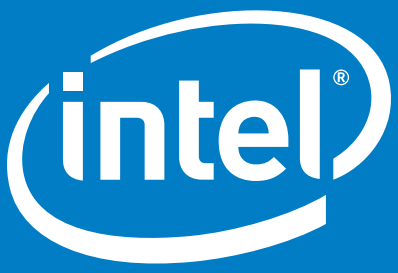
Synapse Mobile Networks*
http://www.intel.com/netcomms/solutions/ipservices-wireless/intel-sun-synapse.pdf

Transitive
http://www.transitive.com/

Using and Redistributing Sun Studio Libraries in an Application
http://docs.sun.com/source/820-5191/stdlibdistr.html

walkcontext
http://docs.sun.com/app/docs/doc/817-3939/6mjgg7hd9?l=en&a=view&q=walkcontext

for more information visit
http://www.intel.com/sunalliance



for more information visit

http://www.intel.com/sunalliance