Guide to SGI™ Compilers and Compiling Tools

007–4479–001

Cover design by Sarah Bolles, Sarah Bolles Design, and Dany Galgani, SGI Technical Publications.

Record of Revision

Version	Description
001	November 2001 Original publication

Contents

About This Guide

This publication describes the documentation available for the SGI compilers, and the SGI compiling performance tools. It provides pointers to that documentation, and gives details about the content of the documentation that you can use while you are using SGI's Fortran and C/C++ compilers.

Related Publications

The following is a list of the documents discussed in this book:

Fortran documentation:

- *MIPSpro Fortran Language Reference Manual, Volume 1*
- *MIPSpro Fortran Language Reference Manual, Volume 2*
- *MIPSpro Fortran Language Reference Manual, Volume 3*
- *MIPSpro Fortran 90 Commands and Directives Reference Manual*
- *MIPSpro Fortran 77 Programmer's Guide*
- *MIPSpro Fortran 77 Language Reference Manual*

C/C++ documentation:

- *C Language Reference Manual*
- *C++ Programmer's Guide*
- *MIPSpro C and C++ Pragmas*

Other compilers:

- *MIPSpro Assembly Language Programmer's Guide*

Optimization, porting, and performance tuning:

- *MIPSpro N32/64 Compiling and Performance Tuning Guide*
- *MIPS O32 Compiling and Performance Tuning Guide*
- *Origin 2000 and Onyx2 Performance Tuning and Optimization Guide*

- *MIPSpro 64-Bit Porting and Transition Guide*
- *MIPSpro N32 ABI Handbook*
- *Application Programmer's I/O Guide*

Debugging tools:

- *dbx User's Guide*
- *dbx Quick Reference Card*
- *ProDev Workshop: Debugger User's Guide*

Performance analysis tools:

- *ProDev Workshop: Performance Analyzer User's Guide*
- *ProDev WorkShop: ProMP User's Guide*
- *ProDev Workshop: Tester User's Guide*
- *ProDev Workshop: Static Analyzer User's Guide*
- *SpeedShop User's Guide*

In addition to these books which document current compilers and tools, other books document older versions of SGI's products. These older books are mentioned in the appropriate chapters.

Obtaining Publications

To obtain SGI documentation, go to the SGI Technical Publications Library at:

`http://techpubs.sgi.com.`

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.
GUI	This font denotes the names of graphical user interface (GUI) elements such as windows, screens, dialog boxes, menus, toolbars, icons, buttons, boxes, fields, and lists.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number is located at the bottom of each page.)

You can contact us in any of the following ways:

- Send e-mail to the following address:

  `techpubs@sgi.com`

- Use the Feedback option on the Technical Publications Library World Wide Web page:

  `http://techpubs.sgi.com`

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  Technical Publications
  SGI
  1600 Amphitheatre Pkwy., M/S 535
  Mountain View, California 94043–1351

- Send a fax to the attention of “Technical Publications” at +1 650 932 0801.

We value your comments and will respond to them promptly.

Introduction

This document is a guide to the SGI compilers, compiling tools, and the documentation for those products. It provides overview information about the compilers and the performance tools used with the compilers. It also provides a brief description of the documentation available for all of these SGI products.

The SGI compilers include the FORTRAN 77 compiler, the Fortran 90 compiler, the C compiler, the C++ compiler, and the Assembler.

Compiling tools include the WorkShop suite of tools (Debugger, Performance Analyzer, Static Analyzer, ProMP, and Tester) as well as `dbx` and SpeedShop. In addition to the compilers mentioned previously, these tools also support the Ada compiler.

This book discusses the following topics:

- Chapter 2, "Compilers and Compiler Documentation", page 5, describes the different SGI compilers and the documentation that accompanies those compilers.
- Chapter 3, "Debuggers and Debugging Documentation", page 9, describes the debugging tools that are available.
- Chapter 4, "Optimization, Porting and Tuning Tools and Documentation", page 11, describes optimization tools and the tuning and porting guides available for the o32, n32 and 64–bit compiler systems.
- Chapter 5, "Performance Analysis Tools and Documentation", page 15, describes the performance tools available with the WorkShop product set and also describes the SpeedShop analysis tool.

There are three “versions” of Fortran and C/C++ compilers in use at SGI:

- The older 32-bit compiler (known as the `o32` compiling system).
- The newer 32-bit compiler (known as the `n32` compiling system).
- The 64-bit compiler.

Chapter 2, "Compilers and Compiler Documentation", page 5, discusses these different compiling systems and the documentation that supports those systems.

Sources of Performance Problems

To tune a program's performance, you must first determine where machine resources are being used. At any point in a process, there is one limiting resource controlling the speed of execution. Processes can be slowed down by:

- CPU speed and availability: a *CPU-bound* process spends its time executing in the CPU and is limited by CPU speed and availability. To improve the performance of CPU-bound processes, you may need to streamline your code. This can entail modifying algorithms, reordering code to avoid interlocks, removing nonessential steps, blocking to keep data in cache and registers, or using alternative algorithms.
- I/O processing: an *I/O-bound* process has to wait for input/output (I/O) to complete. I/O may be limited by disk access speeds or memory caching. To improve the performance of I/O-bound processes, you can try one of the following techniques:
  - Improve overlap of I/O with computation
  - Optimize data usage to minimize disk access
  - Use data compression
- Memory size and availability: a program that continuously needs to swap out pages of memory is called *memory-bound*. Page thrashing is often due to accessing virtual memory on a haphazard rather than strategic basis; cache misses result. Insufficient memory bandwidth could also be the problem.

  To fix a memory-bound process, you can try to improve the memory reference patterns or, if possible, decrease the memory used by the program.
- Bugs: you may find that a bug is causing the performance problem. For example, you may find that you are reading in the same file twice in different parts of the program, that floating-point exceptions are slowing down your program, that old code has not been completely removed, or that you are leaking memory (making `malloc` calls without the corresponding calls to `free`).
- Performance phases: because programs exhibit different behavior during different phases of operation, you need to identify the limiting resource during each phase. A program can be I/O-bound while it reads in data, CPU-bound while it performs computation, and I/O-bound again in its final stage while it writes out data. Once you've identified the limiting resource in a phase, you can perform an in-depth analysis to find the problem. And after you have solved that problem, you can

check for other problems within the phase. Performance analysis is an iterative process.

The documentation available for the compilers and the performance tools can help you pinpoint where these problems are occuring, and can help you determine how to make the necessary changes to improve program performance.

Compilers and Compiler Documentation

SGI supports several compiler and programming languages. This chapter discusses the different compilers and the documentation that supports those compilers. It covers the following topics:

- "Fortran Compilers"
- "C and C++ Compilers", page 7
- "Other Compilers", page 8

All documentation mentioned in this chapter is available online or can be ordered. See the SGI Technical Publications Library at `http://techpubs.sgi.com` for details.

Fortran Compilers

SGI provides support for the FORTRAN 77 and Fortran 90 programming languages, as well as provides directives and extensions to each language.

There are two versions of each compiler, as discussed in the following subsections:

- "MIPSpro FORTRAN 77 and MIPSpro Fortran 90"
- "Fortran 77 and Fortran 90"

See these subsections for details about the documentation which supports each version of the compiler.

MIPSpro FORTRAN 77 and MIPSpro Fortran 90

MIPSpro FORTRAN 77 and MIPSpro Fortran 90 (sometimes called "MIPSpro 7 Fortran 90") have been available on SGI systems since the mid-1990s. These compilers support the newer `-n32` and `-64` Application Binary Interface (ABI).

The following documentation is used with the Fortran 90 compiler:

- The *MIPSpro Fortran 90 Commands and Directives Reference Manual* contains information about the command line options, the directives recognized by the compiler, the OpenMP API multiprocessing directives, source preprocessing, and the Auto-Parallelizing Option.

The following books describe the Fortran 90 language as implemented by SGI's MIPSpro Fortran 90 compiler.

- The *MIPSpro Fortran Language Reference Manual, Volume 1*. Chapters 1 through 8 correspond to sections 1 through 8 of the Fortran standard.
- The *MIPSpro Fortran Language Reference Manual, Volume 2*. Chapter 1 through 6 correspond to sections 9 through 14 of the standard.
- The *MIPSpro Fortran Language Reference Manual, Volume 3*. This manual contains compiler information that supplements the standard. This volume also contains the complete Fortran syntax in Backus-Naur form (BNF).

The following books describe the FORTRAN 77 language as implemented by the MIPSpro Fortran 77 compiler:

- The *MIPSpro Fortran 77 Programmer's Guide* describes SGI's implementation of the FORTRAN 77 standard. This book contains information about compiling, linking, and running programs; system functions and subroutines; and Fortran program interfaces.
- The *MIPSpro Fortran 77 Language Reference Manual* describes the Fortran character set, constants and data structures, specification statements, assignment and data statements, and other elements of the FORTRAN 77 programming language.

In addition to these books, many man pages document the compilers, the libraries used with the compilers, and several of the tools used by the compilers (including `f77`(1), `f90`(1), `f77.f90.difs`(1), and `intro_intrin`(3i)). These man pages are shipped as part of the software and are available online using the `man` command.

Fortran 77 and Fortran 90

Prior to 1996, SGI provided support for the Fortran 77 and Fortran 90 programming languages. The older version of the Fortran 77 compiler supports only the older `o32` ABI.. The older version of the Fortran 90 compiler supported the n32 and 64–bit ABI, but that compiler is based on older technology than the current MIPSpro Fotran 90 compiler.

The *Fortran 77 Programmer's Guide* and *Fortran 77 Language Reference Manual* document the older versions of the Fortran 77 compiler. The *MIPSpro Fortran 90 Programmer's Guide* documents the older Fortran 90 compiler.

Note: Documentation for the older compilers is in maintenance, and should not be considered current.

In addition to these books, the *MIPS O32 Compiling and Performance Tuning Guide* discusses tuning issues in the older `o32` compilers.

C and C++ Compilers

SGI provides support for the C and the C++ programming language. Like the Fortran programming language, SGI has different versions of C and C++:

- MIPSpro 32–bit version (invoked with `cc -n32`) and the MIPSpro 64–bit version (invoked with `cc -64`).
- An older, "ucode" version of the compiler invoked with `cc -o32`.

The N32 and 64–bit compilers accept a dialect of C++ that closely resembles the ANSI/ISO draft C++ standard. The 32–bit ucode compiler is still available, but it is no longer being enhanced. It is available to support legacy code.

The C++ compiler based on `Cfront` is still available but is no longer supported.

The following books document the newer versions of the C and C++ compilers:

- The *C++ Programmer's Guide* describes SGI's implementation of the C++ standard. It discusses compiling and linking programs; dialect support; how to use templates; and how to transition from the older `Cfront` compiler.
- *MIPSpro C and C++ Pragmas* describes all of the `#pragma` directives that are supported (including automatic parallelization, DSM optimization, inlining, loader, LNO, multiprocessing, precompiled header, scalar optimization, and warning suppression control pragmas).
- The *C Language Reference Manual* contains a summary of the syntax and semantics of the C language as implemented at SGI. It contains an overview of ANSI C, descriptions of lexical conventions, a discussion of operator conversions, and other topics covered in the C standard.
- The *Standard Template Library Programmer's Guide* contains information about STL. This documentation is available online at `http://www.sgi.com/Technology/stl`.

In addition to these books, several man pages document the compilers and the compiler libraries (including `cc`(1) and `cpp`(1)).

Other Compilers

SGI provides support for several other programming languages:

- Assembler, which is documented in the *MIPSpro Assembly Language Programmer's Guide*.
- The *Pascal Programming Guide*

In addition to these books, several man pages document these products (including the `as`(1) man page).

The *MIPSpro N32/64 Compiling and Performance Tuning Guide* discusses writing assembly code for the N32 and 64–bit systems.

Many of the optimization and performance tools also support the Ada programming language.

Debuggers and Debugging Documentation

This chapter discusses the debugging tools available on SGI systems. It contains the following topics:

All documentation mentioned in this chapter is available online or can be ordered. See the SGI Technical Publications Library at `http://techpubs.sgi.com` for details.

dbx

`dbx` is a source level debugger used to debug programs in C, C++, Fortran, and assembly language. You can use `dbx` to trace problems in a program at the source code level, rather than at the machine code level. `dbx` enables you to control a program's execution, symbolically monitoring program control flow, variables, and memory locations. You can also use `dbx` to trace the logic and flow of control to acquaint yourself with a program written by someone else.

`dbx` provides a robust command language; using that language, you can perform the following functions:

- examine core dumps
- display and change program variables
- determine variable scope
- control program execution
- debug machine language code
- debug multiprocessed programs

`dbx` is documented in the *dbx User's Guide*. The *dbx Quick Reference Card* is also available.

ProDev™ WorkShop Debugger

The ProDev WorkShop Debugger is a UNIX source-level debugging tool for SGI MIPS systems. It displays program data and execution status in real time; it can be used to debug Ada, C, C++, FORTRAN 77, and Fortran 90 programs.

The Debugger lets you set various types of breakpoints and watchpoints where you can view data (such as variables, expressions, arrays, etc.) You can use the **Views** menu to inspect the following types of data:

- **Call Stack**, to inspect the call stack at the breakpoints.
- **Expression View**, to inspect the value of specified expressions.
- **Variable Browser**, to inspect the values, types, or addresses of variables.
- **Structure Browser**, to inspect data structures.
- **Multiprocess View**, to inspect the values of multiple and / or pthreaded processes.
- **Array Browser**, to inspect the values of an array variable.
- **Memory View**, to inspect the values in specified memory locations.
- **Register View**, to inspect registers.
- **Disassembly View**, to inspect the disassembled code.

The ProDev WorkShop Debugger also includes **Fix+Continue**, which gives you the ability to make changes to a program written in C or C++ without having to recompile and link the entire program before continuing to debug the code. With Fix+Continue, you can edit a function, parse the new functions, and continue execution of the program being debugged.

The **X/Motif Analyzer** in the Debugger provides specific debugging support for X/Motif applications. You can issue X/Motif analyzer commands graphically from the X/Motif analyzer subwindow of the Debugger Main Window.

The Debugger is documented in *ProDev Workshop: Debugger User's Guide*.

Optimization, Porting and Tuning Tools and Documentation

Several different books document how to port code from older versions of compilers to newer versions, how to tune programs for best use, and how to optimize code for use on different hardware systems. The books discussed in this chapter focus primarily on those topics.

In addition to the books discussed in this chapter, the books for each compiling system also contain information about optimization and tuning. You may wish to check those books first for information that is specific to your compiling system before checking the books mentioned here, which are more general in nature.

Optimization and *tuning* are terms that are often used interchangeably. As used in this document, both terms mean a focus on exploiting the features of hardware and software to extract the best performance from the system and the code.

The topic of porting code from older systems is often combined with information about how to fine-tune that code for the newer systems. Thus, "porting and tuning" guides often also discuss some optimization topics in the course of discussing tuning.

All documentation mentioned in this chapter is available online or can be ordered. See the SGI Technical Publications Library at `http://techpubs.sgi.com` for details.

Optimization Guides

The following books discusses optimization topics in detail (but do not discuss porting code in any detail):

- The *Origin 2000 and Onyx2 Performance Tuning and Optimization Guide* describes tuning and optimization in the context of specific hardware architectures; it includes the following topics:
  - details about the SNO hardware architecture
  - SNO memory management
  - tuning for a single process
  - using basic compiler optimizations

  - optimizing cache utilization
  - tuning for parallel processing
- The *Application Programmer's I/O Guide* discusses features that can affect I/O in your Fortran program; it includes the following topics:
  - Fortran I/O extensions provided by SGI
  - the `assign` environment
  - file structures
  - using FFIO (Flexible File I/O)
  - I/O optimizaton

Porting and Tuning Guides

The following books focus on porting code from older systems and also focus on tuning issues for that code:

- *MIPSpro N32/64 Compiling and Performance Tuning Guide* (formerly entitled *MIPSpro Compiling, Debugging, and Performance Tuning Guide*) describes the components of the MIPSpro compiling systems and includes the following topics:
  - how to use dynamic shared objects (DSOs)
  - how to use interprocedural analysis (IPA) and loop nest optimization (LNO) for optimization
  - how to code 64–bit programs
  - how to port code from the older (ucode) 32–bit mode to the N32 and and 64–bit modes
- The *MIPSpro N32 ABI Handbook* describes the 32–bit Application Binary Interface (ABI) and includes the following topics:
  - ABI overview
  - calling convention implementations
  - N32 compatibility

  - porting issues
  - assembly language programming issues
- The *MIPS O32 Compiling and Performance Tuning Guide* describes how to use tools to optimize older O32 code; it includes the following topics:
  - how to use dynamic shared objects (DSOs)
  - how to use performance tools for profiling and pc sampling
  - how to optimize performance with object libraries and compiler options

Performance Analysis Tools and Documentation

Several different products are available to help you analyze your program's code and determine where optimization techniques can be applied. Many of these products are in the ProDev™ WorkShop suite of tools.

This chapter discusses the following topics:

All documentation mentioned in this chapter is available online or can be ordered. See the SGI Technical Publications Library at `http://techpubs.sgi.com` for details. In addition, the `workshop`(1) man page lists all man pages used with the performance tools and provides a summary of each performance tool product.

ProDev™ WorkShop Performance Analyzer

You can use the ProDev WorkShop Performance Analyzer to check your program for different performance problems. This tool has three major windows that display performance information:

- The function list area, which shows functions and their performance metrics.
- The system resource usage chart, which shows the mode of the program at any time.
- The time line, which shows when sample events occur in experiments and controls the scope of analysis.

To conduct performance analysis, first run an experiment to collect performance data. You can specify the objective of your experiment through a task menu or with the SpeedShop `ssrun`(1) command. The Performance Analyzer reads the required data and provides charts, tables, and annotated code to help you analyze the results.

There are three general techniques for collecting performance data:

- Counting: this involves counting the exact number of times each function or basic block has been executed.
- Profiling: the program's program counter (PC), call stack, and/or resource consumption are periodically examined and recorded.
- Tracing: events that impact performance, such as reads and writes, system calls, floating-point exceptions, and memory allocations, reallocations, and frees, can be traced.

The ProDev WorkShop Performance Analyzer is documented in the *ProDev Workshop: Performance Analyzer User's Guide*.

ProDev™ WorkShop ProMP

ProMP is a companion product to the WorkShop suite of tools. It is used to analyze programs that have been parallelized. It is integrated with the other WorkShop tools to let you examine a program's loops in conjunction with a performance experiment on either a single processor or a multiprocessor run.

Before using ProMP, you must first compile your program with the appropriate auto-parallelizing options. The compiler then generates its output files and an analysis file, which ProMP then reads and analyzes.

The ProMP documentation includes several tutorials to help you learn to use the product; these tutorials cover the following topics:

- compiling a program for ProMP use
- viewing detailed information about code and loops
- examining loops with obstacles to parallelization
- examining nested loops
- modifying source files and recompiling the code
- using OpenMP directives
- using ProMP with performance data

The ProMP product is documented in the *ProDev WorkShop: ProMP User's Guide*.

ProDev™ Workshop Tester

ProDev WorkShop Tester is a quality assurance tool for test coverage over sets of tests. This product is used by software and test engineers and others involved in the development, testing, and maintenance of software projects.

WorkShop Tester has the following features:

- it provides visualization of coverage data
- it provides useful measures of test coverage over a set of tests/experiments
- it allows you to view the coverage results of a dynamically shared object (DSO) by executables that use it
- it provides a comparison of coverage over different program versions
- it provides tracing capabilities for function arcs that go beyond traditional test coverage tools

There are two versions of Tester: `cvcov` is the command line version of the test coverage program and `cvxcov` is the GUI version of the test coverage program.

Most of the functionality is available from either program, although the graphical representations of the data are available only from `cvxcov`, the GUI tool.

The Tester product is documented in the *ProDev Workshop: Tester User's Guide*.

ProDev™ WorkShop Static Analyzer

The Static Analyzer is the WorkShop tool for examining the structure of a program's source code and the relationships between its parts, such as files, functions, and variables.

The Static Analyzer shows code structure, including how functions within programs call each other, where and how variables are defined, how files depend on each other, where macros are placed, and other structural details. The Static Analyzer is interactive, so you can quickly locate the portion of code structure that interests you, or you can step back for an overview. Because the Static Analyzer recognizes the connections between elements of the source code, you can readily trace how a proposed change to one element will affect related elements.

The Static Analyzer provides two modes for extracting static analysis data from your source files:

- Scanner mode: a fast, general-purpose scanner that looks through code with minimal sensitivity to the programming language. Scanner mode does not require that your code compile.
- Parser mode: a language-sensitive scanner that can be run at compile time by setting a switch.

The Static Analyzer is documented in the *ProDev Workshop: Static Analyzer User's Guide*.

SpeedShop

The SpeedShop set of tools help you measure program performance using SpeedShop commands. These tools allow you to run experiments and generate reports that track down the sources of performance problems.

SpeedShop consists of a set of commands that can be run in a shell, an application programming interface (API) to provide some control over data collection, and a number of libraries to support the commands.

The following SpeedShop commands help you analyze your programs:

- `ssusage`: Collects information about your program's use of machine resources. Output from `ssusage` can be used to determine where most resources are being spent.
- `ssrun`: Allows you to run experiments on a program to collect performance data. It establishes the environment to capture performance data for an executable, creates a process from the executable (or from an instrumented version of the executable) and runs it.
- `prof`: Analyzes the performance data recorded with `ssrun` and provides formatted reports.

The following additional commands are also provided:

- `squeeze`: Allocates a region of virtual memory and locks the virtual memory down into real memory, making it unavailable to other processes.
- `thrash`: Allows you to allocate a block of memory, then access the allocated memory to explore system paging behavior.

The SpeedShop product is documented in the *SpeedShop User's Guide*.

Index