



Fortran User's Guide

FORTRAN 77 5.0 — Fortran 90 2.0

901 San Antonio Road
Palo Alto, , CA 94303-4900
USA 650 960-1300 fax 650 969-9131

Part No: 805-4941
Revision A, February 1999

Copyright Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

All rights reserved. This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX[®] system, licensed from Novell, Inc., and from the Berkeley 4.3 BSD system, licensed from the University of California. UNIX is a registered trademark in the United States and in other countries and is exclusively licensed by X/Open Company Ltd. Third-party software, including font technology in this product, is protected by copyright and licensed from Sun's suppliers. RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

Sun, Sun Microsystems, the Sun logo, SunDocs, SunExpress, Solaris, Sun Performance Library, Sun Performance WorkShop Fortran, Sun Visual WorkShop C++, Sun WorkShop, and Sun WorkShop Professional are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and in other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK[®] and Sun[™] Graphical User Interfaces were developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox Corporation in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a nonexclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

f90 is derived from Cray CF90[™], a product of Silicon Graphics, Inc.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

Copyright 1999 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie et la décompilation. Aucune partie de ce produit ou de sa documentation associée ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Des parties de ce produit pourront être dérivées du système UNIX[®] licencié par Novell, Inc. et du système Berkeley 4.3 BSD licencié par l'Université de Californie. UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays, et licenciée exclusivement par X/Open Company Ltd. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, SunDocs, SunExpress, Solaris, Sun Performance Library, Sun Performance WorkShop Fortran, Sun Visual WorkShop C++, Sun WorkShop, et Sun WorkShop Professional sont des marques déposées ou enregistrées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC, utilisées sous licence, sont des marques déposées ou enregistrées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Les utilisateurs d'interfaces graphiques OPEN LOOK[®] et Sun[™] ont été développés de Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox Corporation pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique, cette licence couvrant aussi les licenciés de Sun qui mettent en place les utilisateurs d'interfaces graphiques OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

f90 est dérivé de CRAY CF90[™], un produit de Silicon Graphics, Inc.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A REpondre A UNE UTILISATION PARTICULIERE OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.



Contents

- Preface** xiii
- 1. Introduction** 1-1
 - Standards Conformance 1-1
 - Features of the Fortran Compilers 1-2
 - Other Fortran Utilities 1-3
 - Debugging Utilities 1-3
 - Sun Performance Library™ 1-4
 - Man Pages 1-4
 - READMEs 1-5
 - Command-Line Help 1-6
- 2. Using Sun Fortran Compilers** 2-1
 - A Quick Start 2-2
 - Invoking the Compiler 2-3
 - Compile-Link Sequence 2-3
 - Command-Line File Name Conventions 2-4
 - Source Files 2-5
 - Source File Preprocessors 2-5
 - Separate Compiling and Linking 2-5
 - Consistent Compiling and Linking 2-6

	Linking Mixed Fortran 90 and Fortran 77 Compilations	2-6
	Unrecognized Command-Line Arguments	2-6
	Modules (Fortran 90)	2-7
	Directives	2-7
	General Directives	2-8
	Parallelization Directives	2-11
	Compiler Usage Tips	2-12
	Determining Platform Hardware	2-12
	Simplifying Options	2-12
	Memory Size	2-13
3.	f77/E90 Compiler Options	3-1
	Command Syntax	3-1
	Options Syntax	3-2
	Options Summaries	3-3
	Commonly Used Options	3-3
	Debugging Options	3-4
	Floating-Point Options	3-5
	Library Options	3-6
	Licensing Options	3-7
	Performance Options	3-8
	Parallelization Options	3-10
	Profiling Options	3-11
	Alignment Options	3-11
	Backward Compatibility and Legacy Options	3-12
	Obsolescent Options	3-13
	Options Reference	3-13
	-386	3-14
	-486	3-14

-a 3-14
-ansi 3-15
-arg=local 3-15
-autopar 3-16
-B{static|dynamic} 3-17
-C 3-18
-c 3-18
-cg89 3-18
-cg92 3-18
-copyargs 3-19
-Dname[=*def*] 3-19
-dalign 3-20
-dbl 3-21
-dbl_align_all=y 3-22
-depend 3-22
-dryrun 3-23
-d{y|n} 3-23
-e 3-23
-erroff=*taglist* 3-24
-errtags 3-24
-explicitpar 3-24
-ext_names=*e* 3-25
-F 3-26
-f 3-26
-fast 3-27
-fixed 3-28
-flags 3-28
-fnonstd 3-29

-fns[={no|yes}] 3-29
-fpovery[={yes|no}] 3-30
-fprecision=*p* 3-31
-free 3-31
-fround=*r* 3-31
-fsimple[=*n*] 3-32
-fstore 3-33
-ftrap=*t* 3-33
-G 3-34
-g 3-34
-hnm 3-35
-help 3-35
-I*dir* 3-35
-i2 3-36
-i4 3-36
-inline=*f1*[,...*fn*] 3-37
-Kpic 3-37
-KPIC 3-38
-L*dir* 3-38
-lx 3-38
-libmil 3-39
-loopinfo 3-39
-M*dir* 3-40
-misalign 3-40
-mp={sun|cray} 3-41
-mt 3-41
-native 3-42
-noautopar 3-42

-nodepend 3-42
-noexplicitpar 3-42
-nofstore 3-43
-nolib 3-43
-nolibmil 3-44
-noqueue 3-44
-noreduction 3-44
-norunpath 3-44
-O[*n*] 3-45
-O 3-45
-O1 3-46
-O2 3-46
-O3 3-46
-O4 3-46
-O5 3-46
-o *nm* 3-47
-oldldo 3-47
-onetrip 3-47
-p 3-47
-pad[=*p*] 3-48
-parallel 3-49
-pentium 3-50
-pg 3-50
-pic 3-50
-PIC 3-51
-Qoption *pr ls* 3-52
-qp 3-52
-R *ls* 3-52

-r8 3-53
-reduction 3-54
-S 3-55
-s 3-55
-sb 3-55
-sbfast 3-55
-silent 3-55
-stackvar 3-56
-stop_status=*yn* 3-57
-temp=*dir* 3-58
-time 3-58
-U 3-58
-u 3-59
-unroll=*n* 3-59
-V 3-59
-v 3-60
-vax=*v* 3-60
-vpara 3-61
-w 3-61
-xa 3-62
-xarch=*a* 3-62
-xautopar 3-65
-xcache=*c* 3-65
-xcg89 3-66
-xcg92 3-66
-xchip=*c* 3-66
-xcode=*code* 3-68
-xcommonchk[={no|yes}] 3-68

-xcrossfile [=n] 3-69
-xdepend 3-69
-xexplicitpar 3-70
-xF 3-70
-xhelp=*h* 3-70
-xildoff 3-70
-xildon 3-71
-xinline=*f1*[,...,*fn*] 3-71
-xl[d] 3-71
-xlibmil 3-72
-xlibmopt 3-72
-xlic_lib=*libs* 3-72
-xlicinfo 3-73
-Xlist[*x*] 3-73
-xloopinfo 3-75
-xmaxopt[=*n*] 3-75
-xnolib 3-75
-xnolibmil 3-76
-xnolibmopt 3-76
-xO[*n*] 3-76
-xpad 3-76
-xparallel 3-76
-xpg 3-76
-xpp={fpp|cpp} 3-77
-xprefetch[={yes|no}] 3-77
-xprofile=*p* 3-77
-xreduction 3-78
-xregs=*r* 3-78

-xs 3-79
-xsafe=mem 3-80
-xsb 3-80
-xsbfast 3-80
-xspace 3-80
-xtarget=*t* 3-81
-xtime 3-81
-xtypemap=*spec* 3-82
-xunroll=*n* 3-83
-xvector[={yes|no}] 3-83
-xvpara 3-83
-zlp 3-83
-ztext 3-84

A. Runtime Error Messages A-1

Operating System Error Messages A-1

Signal Handler Error Messages A-1

I/O Error Messages (f77) A-2

I/O Error Messages (f90) A-6

B. Features Release History B-1

f77 New Features and Changes B-1

Features in f77 5.0 That are New Since 4.2 B-1

Features in f77 4.2 That are New Since 4.0 B-2

Features in f77 4.0 that are New Since 3.0/3.0.1 B-3

FORTRAN 77 Upward Compatibility B-5

Fortran 3.0/3.0.1 to 4.0 B-5

BCP: Running Applications from Solaris 1 in 2 B-5

f90 New Features and Changes B-5

New Features in f90 2.0 Since 1.2: B-5

C.	Fortran 90 Features and Differences	C-1
	Features	C-1
	Tabs in the Source	C-1
	Continuation Line Limits	C-2
	Fixed-Form Source Lines	C-2
	Directives	C-2
	Tab Form	C-2
	Source Form Assumed	C-3
	Boolean Type	C-4
	Abbreviated Size Notation for Numeric Data Types	C-7
	Cray Pointers	C-8
	Cray Character Pointers	C-13
	Intrinsics	C-14
	Directives	C-16
	Form of General Directive Lines	C-16
	FIXED and FREE Directives	C-17
	Parallelization Directives	C-18
	Form of Parallelization Directive Lines	C-18
	Compatibility with FORTRAN 77	C-19
	Source	C-19
	Linking with f77-Compiled Routines	C-20
	I/O	C-20
	Intrinsics	C-21
	Forward Compatibility	C-22
	Mixing Languages	C-22
	Module Files	C-22
D.	-xtarget Platform Expansions	D-1
	Index	9

Preface

This guide describes the compile-time environment and command-line options for the two Sun[™] Fortran compilers: f77 (FORTRAN 77 version 5.0) and f90 (Fortran 90 version 2.0). Runtime error messages and new features of the compilers are listed in appendixes.

Discussion of Fortran programming issues on Solaris[™] operating environments, including input/output, application development, library creating and use, program analysis, porting, optimization, and parallelization can be found in the companion *Sun Fortran Programming Guide*.

Note - This guide covers the Sun FORTRAN 77 and Fortran 90 compilers. The text uses "f77/f90" and "Fortran" to indicate information that is common to *both* compilers.

Who Should Use This Book

This guide is intended for scientists, engineers, and programmers who have a working knowledge of the Fortran language and wish to learn how to use the Sun Fortran compilers effectively. Familiarity with the Solaris[™] operating environment or UNIX[®] in general is also assumed.

How This Book Is Organized

This guide is organized into the following chapters and appendixes:

- Chapter 1, "*Introduction*," briefly describes the features of the compilers.
- Chapter 2, "*Using Sun Fortran Compilers*," discusses the compiler environments.
- Chapter 3, "*Sun Fortran Compiler Options*," gives detailed descriptions of all the compile-time command-line options and flags.
- Appendix A, "*Runtime Error Messages*," lists error messages issued by the Fortran runtime library and operating environment.
- Appendix B, "*Feature Release History*," notes new features of the compilers and changes in recent releases.
- Appendix C, "*Fortran 90 Features and Differences*," describes the differences between the Sun f90 compiler and the Fortran 90 standard.
- Appendix D, "*-xtarget Platform Extensions*," lists all the platform system names accepted by the compiler `-xtarget` option.

Multiplatform Release

Note - The name of the latest Solaris operating environment release is Solaris 7 but some documentation and path or package path names may still use Solaris 2.7 or SunOS 5.7.

The Sun Fortran documentation covers the release of the Fortran compilers on a number of operating environments and hardware platforms:

FORTRAN 77 5.0 is released for:

- Solaris 2.5.1, 2.6, and Solaris 7 environments on:
 - architectures based on the SPARC[™] microprocessor
 - x86-based architectures, where x86 refers to the Intel[®]™ implementation of one of the following: Intel 80386[™], Intel 80486[™], Pentium[™], or the equivalent

Fortran 90 2.0 is released for:

- Solaris 2.5.1, 2.6, and Solaris 7 environments on SPARC processors only.

Note - The term "x86" refers to the Intel 8086 family of microprocessor chips, including the Pentium, Pentium Pro, and Pentium II processors and compatible microprocessor chips made by AMD and Cyrix. In this document, the term "x86" refers to the overall platform architecture. Features described in this book that are particular to a specific platform are differentiated by the terms "SPARC" and "x86" in the text.

Related Books

The following books augment this manual and provide essential information:

- *Fortran Programming Guide*. Discusses issues relating to input/output, libraries, program analysis, debugging, and performance.
- *FORTRAN 77 Language Reference*. Complete language reference.
- *Fortran Library Reference*—gives details on the language and routines.
- *Sun Performance WorkShop Fortran Overview* gives a high-level outline of the Fortran package suite.

Other Programming Books

- *C User's Guide*—describes compiler options, pragmas, and more.
- *Numerical Computation Guide*—details floating-point computation and numerical accuracy issues.
- *Sun WorkShop Performance Library Reference*—discusses the library of subroutines and functions to perform useful operations in computational linear algebra and Fourier transforms.

Other Sun WorkShop Books

- *Sun WorkShop Quick Install*—provides installation instructions.
- *Sun WorkShop Installation Reference*—provides supporting installation and licensing information.
- *Sun Visual WorkShop C++ Overview*—gives a high-level outline of the C++ package suite.
- *Using Sun WorkShop*—gives information on performing development operations through Sun WorkShop.
- *Debugging a Program With dbx*—provides information on using dbx commands to debug a program.
- *Analyzing Program Performance with Sun WorkShop*—describes the profiling tools; LoopTool, LoopReport, LockLint utilities; and the Sampling Analyzer to enhance program performance.
- *Sun WorkShop TeamWare User's Guide*—describes how to use the Sun WorkShop TeamWare code management tools.

Solaris Books

The following Solaris manuals and guides provide additional useful information:

- *The Solaris Linker and Libraries Guide*—gives information on linking and libraries.
- *The Solaris Programming Utilities Guide*—provides information for developers about the special built-in programming tools available in the SunOS system.

Ordering Sun Documents

The SunDocsSM 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 using this program.

For a list of documents and how to order them, see the catalog section of the SunExpressTM Internet site at <http://www.sun.com/sunexpress>.

Accessing Sun Documents Online

Sun WorkShop documentation is available online from several sources:

- The `docs.sun.com` Web site
- AnswerBook2TM collections
- HTML documents
- Online help and release notes

Using the `docs.sun.com` Web site

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>.

Accessing AnswerBook2 Collections

The Sun WorkShop documentation is also available using AnswerBook2 software. To access the AnswerBook2 collections, your system administrator must have installed

the AnswerBook2 documents during the installation process (if the documents are not installed, see your system administrator or Chapter 3 of *Sun WorkShop Quick Install* for installation instructions). For information about accessing AnswerBook2 documents, see Chapter 6 of *Sun WorkShop Quick Install*, Solaris installation documentation, or your system administrator.

Note - To access AnswerBook2 documents, Solaris 2.5.1 users must first download AnswerBook2 documentation server software from a Sun Web page. For more information, see Chapter 6 of *Sun WorkShop Quick Install*.

Accessing HTML Documents

The following Sun Workshop documents are available online only in HTML format:

- Tools.h++ Class Library Reference
- Tools.h++ User's Guide
- *Numerical Computation Guide*
- Standard C++ Library User's Guide
- *Standard C++ Class Library Reference*
- *Sun WorkShop Performance Library Reference Manual*
- *Sun WorkShop Visual User's Guide*
- Sun WorkShop Memory Monitor User's Manual

To access these HTML documents:

1. Open the following file through your HTML browser:

install-directory/SUNWspr0/DOC5.0/lib/locale/C/html/index.html

Replace *install-directory* with the name of the directory where your Sun WorkShop software is installed (the default is /opt).

The browser displays an index of the HTML documents for the Sun WorkShop products that are installed.

2. Open a document in the index by clicking the document's title.

Accessing Sun WorkShop Online Help and Release Notes

This release of Sun WorkShop includes an online help system as well as online manuals. To find out more see:

- Online Help. A help system containing extensive task-oriented, context-sensitive help. To access the help, choose Help Help Contents. Help menus are available in all Sun WorkShop windows.
- Release Notes. The Release Notes contain general information about Sun WorkShop and specific information about software limitations and bugs. To access the Release Notes, choose Help Release Notes.
- You can view the latest release information regarding the Fortran compilers by invoking a compiler with the `-xhelp=readme` flag.

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name% You have mail.</code>
AaBbCc123	What you type, contrasted with on-screen computer output	<code>machine_name% su</code> <code>Password:</code>
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm <i>filename</i></code> .
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be root to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	machine_name%
C shell superuser prompt	machine_name#
Bourne shell and Korn shell prompt	\$
Bourne shell and Korn shell superuser prompt	#

Other Conventions Used in This Book

The following conventions appear in the text of this book:

- Code listings and examples appear in boxes:

```
WRITE( *, * ) "Hello world"
```

- The symbol □ stands for a blank space where a blank is significant:

```
□□36.001
```

- FORTRAN 77 examples appear in tab format, while Fortran 90 examples appear in free format. Examples common to both FORTRAN 77 and 90 use tab format except where indicated.
- Uppercase characters are generally used to show Fortran keywords and intrinsics (PRINT), and lowercase or mixed case is used for variables (TbarX).
- The Sun Fortran compilers are referred to by their command names, either f77 or f90. "f77/f90" indicates information that is common to both the FORTRAN 77 and Fortran 90 compilers.

- References to online man pages appear with the topic name and section number. For example, a reference to GETENV will appear as *getenv*(3F), implying that the man command to access this page would be: `man -s 3F getenv`
- System Administrators may install the Sun Fortran compilers and supporting material at: `<install_point>/SUNWspr0/SC5.0/` where `<install_point>` is usually `/opt` for a standard install. This is the location assumed in this book.

Introduction

The Sun Fortran compilers, f77 and f90, described in this book (and the companion book *Sun Fortran Programming Guide*) are available under the Solaris operating environment on the various hardware platforms that Solaris supports. The compilers themselves conform to published Fortran language standards, and provide many extended features, including multiprocessor parallelization, sophisticated optimized code compilation, and mixed C/Fortran language support.

Standards Conformance

- f77 was designed to be compatible with the ANSI X3.9-1978 Fortran standard and the corresponding International Organization for Standardization (ISO) 1539-1980, as well as standards FIPS 69-1, BS 6832, and MIL-STD-1753.
- f90 was designed to be compatible with the ANSI X3.198-1992 standard, and ISO/IEC 1539:1991
- Floating-point arithmetic for both compilers is based on IEEE standard 754-1985, and international standard IEC 60559:1989.
- On SPARC platforms, both compilers provide support for the optimization-exploiting features of SPARC V8, and SPARC V9, including the UltraSPARC™ implementation. These features are defined in the SPARC Architecture Manuals, Version 8 (ISBN 0-13-825001-4), and Version 9 (ISBN 0-13-099227-5), published by Prentice-Hall for SPARC International.
- In this document, "Standard" means conforming to the versions of the standards listed above. "Non-standard" or "Extension" refers to features that go beyond these versions of these standards.

The responsible standards bodies may revise these standards from time to time. The versions of the applicable standards to which these compilers conform may be revised or replaced, resulting in features in future releases of the Sun Fortran compilers that create incompatibilities with earlier releases.

Features of the Fortran Compilers

Sun Fortran compilers provide the following features or extensions:

- f77: Global program checking across routines for consistency of arguments, commons, parameters, and the like.
- *SPARC only*: Support for multiprocessor systems, including automatic and explicit loop parallelization, is integrated tightly with optimization.

Note - Parallelization features of the Fortran compilers are only available with the Sun Performance WorkShop.

- f77: Many VAX/VMS Fortran 5.0 extensions, including:
 - NAMELIST
 - DO WHILE
 - Structures, records, unions, maps
 - Variable format expressions
 - Recursion
 - Pointers
 - Double-precision complex
 - *SPARC*: Quadruple-precision real
 - *SPARC*: Quadruple-precision complex
- Cray-style parallelization directives, including TASK COMMON, with extensions for f90.
- Global, peephole, and potential parallelization optimizations produce high performance applications. Benchmarks show that optimized applications can run significantly faster when compared to unoptimized code.
- Common calling conventions on Solaris systems permit routines written in C or C++ to be combined with Fortran programs.
- Support for 64-bit Solaris 7 environments on UltraSPARC platforms.
- Fortran 95 features in f90 include the attributes PURE and ELEMENTAL, and enhanced forms of MAXVAL and MINVAL.
- Call-by-value, %VAL, implemented in both f77 and f90.

- Interoperability between Fortran 77 and Fortran 90 programs and object binaries.

Other Fortran Utilities

The following utilities provide assistance in the development of software programs in Fortran:

- **asa** — This Solaris utility is a Fortran output filter for printing files that have Fortran carriage-control characters in column one. Use `asa` to transform files formatted with Fortran carriage-control conventions into files formatted according to UNIX line-printer conventions. See `asa(1)`.
- **fsplit** — This utility splits one Fortran file of several routines into several files, each with one routine per file. Use `fsplit` on FORTRAN 77 or Fortran 90 source files. See `fsplit(1)`.
- **gprof** — Profile program run-time performance by procedure. (This utility is available if you do a developer install, rather than an end user install of the Solaris operating environment; it is also included if you install the `SUNWbtool` package.)
- **tcov** — Profile program run-time performance by statement.

Debugging Utilities

The following debugging utilities are available:

- **error** — A utility to merge compiler error messages with the Fortran source file. (This utility is included if you do a developer install, rather than an end user install of Solaris; it is also included if you install the `SUNWbtool` package.)
- **-xlist** — A compiler option to check across routines for consistency of arguments, COMMON blocks, and so on.
- **Sun WorkShopTM** — The Sun WorkShop provides a number of debugging utilities such as `dbx` and a data visualizer, presented within an easy-to-use graphical framework.

Sun Performance LibraryTM

The Sun Performance Library is a library of optimized subroutines and functions for computational linear algebra and Fourier transforms. It is based on the standard libraries BLAS1, BLAS2, BLAS3, LINPACK, LAPACK, FFTPACK, and VFFTPACK.

Each subprogram in the Sun Performance Library performs the same operation and has the same interface as the standard library versions, but is generally much faster and possibly more accurate.

See the `performance_library` README file for details.

Man Pages

On-line manual (`man`) pages provide immediate documentation about a command, function, subroutine, or collection of such things.

Sun WorkShop man pages are located in `/opt/SUNWspro/man/` after a standard install of the products. Add this path to your `MANPATH` environment variable to access these Sun WorkShop man pages.

You can display a man page by running the command:

```
demo% man topic
```

Throughout the Fortran documentation, man page references appear with the topic name and man section number: `f77(1)` is accessed with `man f77`. Other sections, denoted by `ieee_flags(3M)` for example, are accessed using the `-s` option on the `man` command:

```
demo% man -s 3M ieee_flags
```

The following lists man pages of interest to Fortran user:

<code>f77(1)</code> and <code>f90(1)</code>	The Fortran compilers command-line options
<code>asa(1)</code>	Fortran carriage-control print output post-processor
<code>dbx(1)</code>	Command-line interactive debugger

fpp(1)	Fortran source code pre-processor
cpp(1)	C source code pre-processor
fsplit(1)	Pre-processor splits Fortran 77 routines into single files
ieee_flags(3M)	Examine, set, or clear floating-point exception bits
ieee_handler(3M)	Handle floating-point exceptions
matherr(3M)	Math library error handling routine
ild(1)	Incremental link editor for object files
ld(1)	Link editor for object files

READMEs

The `READMEs` directory contains files that describe new features, software incompatibilities, bugs, and information that was discovered after the manuals were printed. The location of this directory depends on where your software was installed.

The `READMEs` in a standard install would appear in: `/opt/SUNWsprow/READMEs/`

TABLE 1-1 READMEs of Interest

README File	Describes...
fortran_77	new and changed features, known limitations, documentation errata for this release of the FORTRAN 77 compiler, <code>f77</code> .
fortran_90	new and changed features, known limitations, documentation errata for this release of the Fortran 90 compiler, <code>f90</code> .
math_libraries	optimized and specialized math libraries available.
profiling_tools	using the performance profiling tools, <code>prof</code> , <code>gprof</code> , and <code>tcov</code> .

TABLE 1-1 READMEs of Interest (continued)

README File	Describes...
runtime_libraries	libraries and executables that can be redistributed under the terms of the End User License.
64bit_Compilers	compiling for 64-bit Solaris 2.7 environments.
fpp_white_paper.ps	fpp, the Fortran pre-processor; this is a reference white paper in PostScript format and can be viewed using imagetool.
performance_library	the Sun Performance Library

The READMEs for all compilers are easily accessed by the `-xhelp=readme` command-line option. For example, the command:

```
f90 -xhelp=readme
```

will display the `fortran_90` README file directly.

Command-Line Help

You can view very brief descriptions of the `f77` and `f90` command line options by invoking the compiler's `-help` option as shown below:

```
%f77 -help -or-  
f90 -help
```

Items within [] are optional. Items within < > are variable parameters. Bar | indicates choice of literal values. For example:
`-someoption[=<yes|no>]` implies `-someoption is`
`-someoption=yes`

```
-a:          Collect data for tcov basic block profiling  
             (old format)  
-ansi:      Report non-ANSI extensions.  
-arg=local: Preserve actual arguments over ENTRY statements  
-autopar:   Enable automatic loop parallelization  
             (requires WorkShop license)  
-Bdynamic:  Allow dynamic linking  
-Bstatic:   Require static linking  
-c:         Compile only - produce .o files, suppress linking  
-C:         Enable runtime subscript range checking  
-cg89:      Generate code for generic SPARC V7 architecture  
-cg92:      Generate code for SPARC V8 architecture  
-copyargs:  Allow assignment to constant arguments  
...etc.
```


Using Sun Fortran Compilers

This chapter describes how to use the Fortran 77 and Fortran 90 compilers.

The principal use of any compiler is to transform a program written in a procedural language like Fortran into a data file that is executable by the target computer hardware. As part of its job, the compiler may also automatically invoke a system linker to generate the executable file.

The Sun Fortran 77 and Fortran 90 compilers can also be used to:

- Generate a parallelized executable file for multiple processors (`-parallel`).
- Analyze program consistency across source files and subroutines and generate a report (`-Xlist`).
- Transform source files into:
 - Relocatable binary (`.o`) files, to be linked later into an executable file or static library (`.a`) file.
 - A dynamic shared library (`.so`) file (`-G`).
- Link or relink files into an executable file.
- Compile an executable file with runtime debugging enabled (`-g`).
- Compile with runtime statement or procedure level profiling (`-pg`).
- Compile an executable file with runtime parallelized loop profiling (`-Zlp`).
- Check source code for ANSI standards conformance (`-ansi`).

A Quick Start

This section provides a quick overview of how to use the Sun Fortran compilers to compile and run Fortran programs. A full reference to command-line options appears in the next chapter.

Note - The command line examples in this chapter primarily show `f77` usages. Except where noted, equivalent usages of `f90` are similarly valid; however, the printed output may be slightly different.

The very basic steps to running a Fortran application involve using an editor to create a Fortran source file with a `.f`, `.for`, `.f90`, `.F`, or `.F90` filename suffix; invoking the compiler to produce an executable; and finally, launching the program into execution by typing the name of the file:

Example: This program displays a message on the screen:

```
demo% cat greetings.f
      PROGRAM GREETINGS
      PRINT *, "Real programmers write Fortran!"
      END
demo% f77 greetings.f
greetings.f:
  MAIN greetings:
demo% a.out
  Real programmers write Fortran!
demo%
```

In the example, `f77` compiles source file `greetings.f` and compiles the executable program onto the file, `a.out`, by default. To launch our program, the name of the executable file, `a.out`, is typed at the command prompt

Traditionally, UNIX compilers write executable output to the default file called `a.out`. It can be awkward to have each compilation write to the same file. Moreover, if such a file already exists, it will be overwritten by the next run of the compiler. Instead, use the `-o` compiler option to explicitly specify the name of the executable output file:

```
demo% f77 -o greetings greetings.f
greetings.f:
MAIN greetings:
demo%
```

In the preceding example, the `-o` option tells the compiler to write the executable code to the file `greetings`. (By convention, executable files usually are given the same name as the main source file, but without an extension.)

Alternatively, the default `a.out` file could be renamed via the `mv` command after each compilation. Either way, run the program by typing the name of the executable file:

```
demo% greetings  
Real programmers write Fortran!  
demo%
```

The next sections of this chapter discuss the conventions used by the `f77` and `f90` commands, compiler source line directives, and other issues concerning the use of these compilers. The next chapter describes the command-line syntax and all the options in detail.

Invoking the Compiler

The syntax of a *simple* compiler command invoked at a shell prompt is:

```
f77 [options] files... invokes the Fortran 77 compiler  
f90 [options] files... invokes the Fortran 90 compiler
```

Here *files...* is one or more Fortran source file names ending in `.f`, `.F`, `.f90`, `.F90`, or `.for`; *options* is one or more of the compiler option flags. (Files with names ending in a `.f90` or `.F90` extension are “free-format” Fortran 90 source files recognized only by the `f90` compiler.)

In the example below, `f90` is used to compile two source files to produce an executable file named `growth` with runtime debugging enabled:

```
demo% f90 -g -o growth growth.f fft.f90
```

Compile-Link Sequence

In the previous example, the compiler will automatically generate the loader object files, `growth.o` and `fft.o`, and then invoke the system linker to create the executable program on the file `growth`.

After compilation, the object files, `growth.o` and `fft.o`, will remain. This convention permits easy relinking and recompilation of files.

If the compilation fails, you will receive a message for each error. No `.o` files are generated for those source files with errors, and no executable program is written

Command-Line File Name Conventions

The suffix extension attached to file names appearing on the command-line determine how the compiler will process the file. File names with a suffix extension other than one of those listed below, or without an extension, are passed to the linker.

TABLE 2-1 File Name Suffixes Recognized by Sun Fortran Compilers

Suffix	Language	Action
<code>.f</code>	Fortran 77 or Fortran 90 fixed-format	Compile Fortran source files, put object files in current directory; default name of object file is that of the source but with <code>.o</code> suffix.
<code>.f90</code>	Fortran 90 free-format	Same action as <code>.f</code> (<i>f90 only</i>)
<code>.for</code>	Fortran 77	Same action as <code>.f</code> .
<code>.F</code>	Fortran 77 or Fortran 90 fixed-format	Apply the Fortran (or C) preprocessor to the Fortran 77 source file before compilation.
<code>.F90</code>	Fortran 90 free-format	Apply the Fortran (or C) preprocessor to the Fortran 90 free-format source file before Fortran compiles it.
<code>.s</code>	Assembler	Assemble source files with the assembler.
<code>.S</code>	Assembler	Apply the C preprocessor to the assembler source file before assembling it.
<code>.i1</code>	Inline expansion	Process template files for inline expansion. The compiler will use templates to expand inline calls to selected routines. (Template files are special assembler files; see the <i>inline(1)</i> man page.)
<code>.o</code>	Object files	Pass object files through to the linker.
<code>.a</code> , <code>.so</code> , <code>.so.n</code>	Libraries	Pass names of libraries to the linker. <code>.a</code> files are static libraries, <code>.so</code> and <code>.so.n</code> files are dynamic libraries.

Fortran 90 free-format is described in Appendix C of this manual.

Source Files

The Fortran compilers will accept multiple source files on the command line. A single source file, also called a *compilation unit*, may contain any number of procedures (main program, subroutine, function, block data, module, and so on). There are advantages for organizing an application with one procedure per file, as there are for gathering procedures that work together into a single file. Some of these are described in the Sun *Fortran Programming Guide*.

Source File Preprocessors

Both `f77` and `f90` support two source file preprocessors, `fpp` and `cpp`. Either can be invoked by the compiler to expand source code “macros” and symbolic definitions prior to compilation. The compilers will use `fpp` by default; the `-xpp=cpp` option changes the default from `fpp` to `cpp`. (See also the discussion of the `-Dname` option).

`fpp` is a source preprocessor specifically for the Fortran language. See the `fpp(1)` man page, and the `fpp` white paper in the `READMEs` directory for details. It is invoked by default by `f77` on files with a `.F` extension, and by `f90` on files with a `.F` or `.F90` extension.

The `cpp` program is the C language preprocessor. See `cpp(1)`. Use of `fpp` over `cpp` is recommended.

Separate Compiling and Linking

You can compile and link in separate steps. The `-c` option compiles source files and generates `.o` object files, but does not create an executable. Without the `-c` option the compiler will invoke the linker. By splitting the compile and link steps in this manner, a complete recompilation is not needed just to fix one file, as shown in the following example:

Compile one file and link with others in separate steps:

```
demo% f77 -c file1.f (Make new object file)
demo% f77 -o prgrm file1.o file2.o file3.o (Make executable file)
```

Be sure that the link step lists *all* the object files needed to make the complete program. If any object files are missing from this step, the link will fail with undefined external reference errors (missing routines).

Consistent Compiling and Linking

Ensuring a consistent choice of compiling and linking options is critical whenever compilation and linking are done in separate steps. Compiling any part of a program with any of the following options requires linking with the same options:

`-a`, `-autopar`, `-BX`, `-dbl`, `-fast`, `-G`, `-Lpath`, `-lname`, `-mt`, `-nolib`, `-norunpath`, `-p`, `-pg`, `-`

Example: Compiling `sbr.f` with `-a` and `smain.f` without it, then linking in separate steps (`-a` invokes `tcov` old-style profiling):

```
demo% f77 -c -a sbr.f
demo% f77 -c smain.f
demo% f77 -a sbr.o smain.o          link step; passes -a to the linker
```

Also, a number of options require that *all* source files be compiled with that option, *including* the link step. These include:

`-autopar`, `-cg92`, `-dx`, `-dalign`, `-explicitpar`, `-f`, `-misalign`, `-native`, `-parallel`, `-pent`

Linking Mixed Fortran 90 and Fortran 77 Compilations

As a general rule, if *any* of the object files that make up a program were compiled with `f90`, then the final link step must be done with `f90`. Use `f77` to produce the executable file only if *none* of the `.o` object files were compiled with `f90`.

Unrecognized Command-Line Arguments

Any arguments on the command-line that the compiler does not recognize are interpreted as being possibly linker options, object program file names, or library names.

The basic distinctions are:

- Unrecognized *options* (with a `-`) generate warnings.
- Unrecognized *non-options* (no `-`) generate no warnings. However, they are passed to the linker and if the linker does not recognize them, they generate linker error messages.

For example:

```
demo% f77 -bit move.f          <-
      -bit is not a recognized
f77 option
f77: Warning: Option -bit passed to ld, if ld is invoked, ignored otherwise
move.f:
```

```
MAIN move:
demo% f77 fast move.f          <- The user meant to type
-fast
move.f:
MAIN move:
ld: fatal: file fast: cannot open file; errno=2
ld: fatal: File processing errors. No output written to a.out
```

Note that in the first example, `-bit` is not recognized by `f77` and the option is passed on to the linker (`ld`), who tries to interpret it. Because single letter `ld` options may be strung together, the linker sees `-bit` as `-b -i -t`, which are all legitimate `ld` options! This may (or may not) be what the user expects, or intended.

In the second example, the user intended to type the `f77/f90` option `-fast` but neglected the leading dash. The compiler again passes the argument to the linker which, in turn, interprets it as a file name.

These examples indicate that extreme care should be observed when composing compiler command lines!

Modules (Fortran 90)

`f90` automatically creates module information files for each `MODULE` declaration encountered in the source files, and searches for modules referenced by a `USE` statement. All the modules appearing in a source file are compiled into a single file with the primary name of the `MODULE` and `.mod` suffix. For example, `f90` generates the module information file `list.mod` for the `MODULE list` unit found on file `mysrc.f90`.

The compiler searches the current directory for module files referenced in `USE` statements. Modules files must be compiled before compiling any source file referencing a `MODULE` in a `USE` statement. Directories can be added to the search path with the `-M` command-line option. However, individual `.mod` files cannot be specified directly on the command line.

Directives

Use a source code *directive*, a form of Fortran comment, to pass specific information to the compiler regarding special optimization or parallelization choices. Compiler directives are also called *pragmas*.

Directives unique to `f90` are described in Appendix C.

Note - Directives are not part of the Fortran standard.

General Directives

The various forms of a Sun general Fortran directive are:

```
C$PRAGMA keyword
C$PRAGMA keyword
( a [ , a
] ) [ , keyword ( a [ , a
] ) ] ,
C$PRAGMA SUN keyword
```

The variable *keyword* identifies the specific directive; the *a*'s are arguments.

The general directives recognized only by f77 are:

- C(...) — The listed arguments are external functions written in C.
- WEAK (*name1*[=*name2*]) — Define weak symbol bindings.
- OPT=*n* — Compile subprogram at specified optimization level *n*.

Other general directives recognized by both f77 and f90 are:

- UNROLL=*n* — Request optimizer to attempt loop unrolling to depth *n*.
- PIPELOOP=*n* — Assert to the optimizer that a loop can be pipelined to a length *n*.

A general directive has the following syntax:

- In column one, any of the comment-indicator characters c, C, !, or *
- The next seven characters are \$PRAGMA, no blanks, any uppercase or lowercase
- In any column, the ! comment-indicator character

Observe the following restrictions:

- After the first eight characters, blanks are ignored, and uppercase and lowercase are equivalent, as in Fortran text.
- Because it is a comment, a directive cannot be continued, but you can have many C\$PRAGMA lines, one after the other, as needed.
- If a comment satisfies the above syntax, it is expected to contain one or more directives recognized by the compiler; if it does not, a warning is issued.
- The C preprocessor, cpp, will expand macro symbol definitions within a comment or directive line; the Fortran preprocessor, fpp, will not expand macros in comment lines, and will recognize legitimate f77 and f90 directives and allows limited substitution outside directive keywords. However, be careful with

directives that require the keyword `SUN`. `cpp` will replace lower-case `sun` with a predefined value. Also, if you define a `cpp` macro `SUN` it may interfere with the `SUN` directive keyword. A general rule might be to spell those pragmas in mixed case if the source will be processed by `cpp` or `fpp`, as in:

```
C$PRAGMA Sun UNROLL=3
```

The C Directive (§ 77)

The `C()` directive specifies that its arguments are external functions written in the C language. It is equivalent to an `EXTERNAL` declaration except that unlike ordinary external names, the Fortran compiler will not append an underscore to these argument names. See the Sun *Fortran Programming Guide* for more details.

The `C()` directive for a particular function should appear before the first reference to that function in each subprogram that contains such a reference.

Example - compiling ABC and XYZ for C:

```
EXTERNAL ABC, XYZ  
C$PRAGMA C(ABC, XYZ)
```

The UNROLL Directive

The `UNROLL` directive requires that you specify `SUN` after `C$PRAGMA`.

The `C$PRAGMA SUN UNROLL=n` directive instructs the compiler to unroll loops *n* times during its optimization pass.

n is a positive integer. The choices are:

- If *n*=1, this directive directs the optimizer *not* to unroll any loops.
- If *n*>1, this directive suggests to the optimizer that it unroll loops *n* times.

If any loops are actually unrolled, the executable file becomes larger. For further information, see the *Fortran Programming Guide* chapter on performance and optimization.

Example - unrolling loops two times:

```
C$PRAGMA SUN UNROLL=2
```

The WEAK Directive (§ 77)

The `WEAK` directive defines a symbol to have less precedence than an earlier definition of the same symbol. This pragma is used mainly in sources files for

building libraries. The linker does not produce an error message if it is unable to resolve a weak symbol.

```
C$PRAGMA WEAK (name1 [=name2])
```

`WEAK (name1)` defines *name1* to be a weak symbol. The linker does not produce an error message if it does not find a definition for *name1*.

`WEAK (name1=name2)` defines *name1* to be a weak symbol and an alias for *name2*.

If your program calls but does not define *name1*, the linker uses the definition from the library. However, if your program defines its own version of *name1*, then the program's definition is used and the weak global definition of *name1* in the library is not used. If the program directly calls *name2*, the definition from library is used; a duplicate definition of *name2* causes an error. See the Solaris *Linker and Libraries Guide* for more information.

The OPT Directive (f77)

The `OPT` directive requires that you specify `SUN` after `C$PRAGMA`.

The `OPT` directive sets the optimization level for a subprogram, overriding the level specified on the compilation command line. The directive must appear immediately before the target subprogram, and only applies to that subprogram. For example:

```
C$PRAGMA SUN OPT=2
      SUBROUTINE smart(a,b,c,d,e)
      ...etc
```

When the above is compiled with an `f77` command that specifies `-O4`, the directive will override this level and compile the subroutine at `-O2`. Unless there is another directive following this routine, the next subprogram will be compiled at `-O4`.

The routine must also be compiled with the `-xmaxopt=n` option for the directive to be recognized. This compiler option specifies a maximum optimization value for `PRAGMA OPT` directives: if a `PRAGMA OPT` specifies an optimization level greater than the `-xmaxopt` level, the `-xmaxopt` level is used.

(SPARC Only) The PIPELOOP=*n* Directive

The `PIPELOOP=n` directive requires that you specify `SUN` after `C$PRAGMA`.

This directive must appear immediately before a `DO` loop. *n* is a positive integer constant, or zero, and asserts to the optimizer a dependency between loop iterations. A value of zero indicates that the loop has no inter-iteration dependencies and can be freely pipelined by the optimizer. A positive *n* value implies that the *I*-th iteration

of the loop has a dependency on the $(I-n)$ -th iteration, and can be pipelined at best for only n iterations at a time.

```
C    We know that the value of K is such that there can be no
C    cross-iteration dependencies (E.g.  $K > N$ )
C$PRAGMA SUN PIPELOOP=0
    DO I=1,N
      A(I)=A(I+K) + D(I)
      B(I)=B(I) + A(I)
    END DO
```

For more information on optimization, see the *Fortran Programming Guide*.

Parallelization Directives

Parallelization directives explicitly request the compiler attempt to parallelize the DO loop that follows the directive. The syntax differs from general directives.

Parallelization directives are only recognized when compilation options `-parallel` or `-explicitpar` are used. (expanded f90 parallelization directives are described in Appendix C; details of Fortran parallelization can be found in the *Fortran Programming Guide*.)

Parallelization directives have the following syntax:

- The first character must be in column one.
- The first character can be any one of `c`, `C`, `*`, or `!`.
- The next four characters are `$PAR`, no blanks, either upper or lower case.
- Next follows the directive keyword and options, separated by blanks. The explicit parallelization directive keywords are:

```
TASKCOMMON, DOALL, DOSERIAL, and DOSERIAL*
```

Each parallelization directive has its own set of optional qualifiers that follow the keyword.

Example: Specifying a loop with a shared variable:

```
C$PAR DOALL SHARED(yvalue)
```

See the *Fortran Programming Guide* for details about parallelization and these directives.

Compiler Usage Tips

The next sections suggest a number of ways to use the Sun Fortran compilers efficiently. A complete compiler options reference follows in the next chapter.

Determining Platform Hardware

Some compiler flags allow the user to tune code generation to a specific set of hardware platform options. The utility command `fpversion` displays the hardware platform specifications for the native processor:

```
demo% fpversion
A SPARC-based CPU is available.
CPU's clock rate appears to be approximately 356.2 MHz.
Kernel says CPU's clock rate is 360.0 MHz.
Kernel says main memory's clock rate is 120.0 MHz.

Sun-4 floating-point controller version 0 found.
An UltraSPARC chip is available.
FPU's frequency appears to be approximately 369.5 MHz.

Use "-xtarget=ultra2i -xcache=16/32/1:2048/64/1" code option.

Hostid = 0x808Z2211.
```

It may take a number of seconds before `fpversion` responds while it dynamically calculates apparent hardware clock rates of the CPU and FPU. (The values printed depend on the load on the system at the moment `fpversion` is called.)

See `fpversion(1)` and the *Numerical Computation Guide* for details.

Simplifying Options

You can simplify complicated compiler commands by defining special shell aliases or using the `$FFLAGS` environment variable.

Using Aliases (C Shell)

Example: Define an alias for a command with frequently used options:

```
demo% alias f77fx "f77 -silent -fast -xlist"
```

Example: Using the alias `f77fx`:

```
demo% f77fx any.f
```

The command `f77fx` is now the same as:

```
f77 -silent -fast -Xlist any.f
```

Using Environment Variables

You can specify options by setting the `FFLAGS` or `OPTIONS` variables.

Either `FFLAGS` or `OPTIONS` can be used explicitly in the command line. When you are using `make` files implicit compilation rules, `FFLAGS` is used automatically by the `make` program.

Example: Set `FFLAGS`: (C Shell)

```
demo% setenv FFLAGS '-silent -fast -Xlist'
```

Example: Use `FFLAGS` explicitly:

```
demo% f77 $FFLAGS any.f
```

When using `make`, if the `FFLAGS` variable is set as above and the makefile's compilation rules are *implicit*, that is, there is no *explicit* `f77/f90` compile line, then invoking `make` will result in a compilation equivalent to:

```
f77 -silent -fast -Xlist files...
```

`make` is a very powerful program development tool that can easily be used with all Sun compilers. See the *make(1)* man page and the *Program Development* chapter in the *Fortran Programming Guide*.

Memory Size

A compilation may need to use a lot of memory. This will depend on the optimization level chosen and the size and complexity of the files being compiled. On SPARC platforms, if the optimizer runs out of memory, it tries to recover by retrying the current procedure at a lower level of optimization and resumes subsequent routines at the original level specified in the `-On` option on the command line.

A workstation should have at least 24 megabytes of memory; 32 megabytes are recommended. Memory usage depends on the size of each procedure, the level of

optimization, the limits set for virtual memory, the size of the disk swap file, and various other parameters.

Compiling a single source file containing many routines could cause the compiler to run out of memory or swap space.

If the compiler runs out of memory, try reducing the level of optimization, or split multiple-routine source files into files with one routine per file, using *fsplit*(1).

Swap Space Limits

The SunOS command, `swap -s`, displays available swap space. See *swap*(1M).

Example: Use the `swap` command:

```
demo% swap -s
total: 40236k bytes allocated + 7280k reserved = 47516k used, 1058708k available
```

To determine the actual real memory:

```
demo% /usr/sbin/dmesg | grep mem
mem = 655360K (0x28000000)
avail mem = 602476544
```

Increasing Swap Space

Use *mkfile*(1M) and *swap*(1M) to increase the size of the swap space on a workstation. You must become superuser to do this. `mkfile` creates a file of a specific size, and `swap -a` adds the file to the system swap space:

```
demo# mkfile -v 90m /home/swapfile
/home/swapfile 94317840 bytes
demo# /usr/sbin/swap -a /home/swapfile
```

Control of Virtual Memory

Compiling very large routines (thousands of lines of code in a single procedure) at optimization level `-O3` or higher may require an unreasonable amount of memory. In such cases, performance of the system may degrade. You can control this by limiting the amount of virtual memory available to a single process.

- In a `sh` shell, use the `ulimit` command. See *sh*(1).

Example: Limit virtual memory to 16 Mbytes:

```
demo$ ulimit -d 16000
```

- In a `cs` shell, use the `limit` command. See `cs(1)`.

Example: Limit virtual memory to 16 Mbytes:

```
demo% limit datasize 16M
```

Each of these command lines causes the optimizer to try to recover at 16 Mbytes of data space.

This limit cannot be greater than the system's total available swap space and, in practice, must be small enough to permit normal use of the system while a large compilation is in progress.

Be sure that no compilation consumes more than half the space.

Example: With 32 Mbytes of swap space, use the following commands:

In a `sh` shell:

```
demo$ ulimit -d 1600
```

In a `cs` shell:

```
demo% limit datasize 16M
```

The best setting depends on the degree of optimization requested, and the amount of real memory and virtual memory available.

In 64-bit Solaris 7 environments, the soft limit for the size of an application data segment is 2 Gbytes. If your application needs to allocate more space, use the shell's `limit` or `ulimit` command to remove the limit. For `cs` use:

```
demo% limit datasize unlimited
```

or for `sh`, `ksh`:

```
demo$ ulimit -d unlimited
```

See the *Solaris 7 64-bit Developer's Guide* for more information.

£77/£90 Compiler Options

This chapter details the command-line options for the Sun £77 and £90 compilers.

- A description of the syntax used for compiler option flags appears on “Options Syntax” on page 3-2
- Summaries of options arranged by functionality starts on “Options Summaries” on page 3-3.
- The complete reference detailing each compiler option flag starts on “Options Reference ” on page 3-13.

Some options are not available on both platforms (SPARC or x86) or compilers (£77 or £90). Check the index list and reference section for availability.

Command Syntax

The general syntax of the compiler command line is:

```
£77 [options] list_of_files [-lx]  
£90 [options] list_of_files [-lx]
```

Items in square brackets indicate optional parameters. The brackets are not part of the command. The *options* are a list of option keywords prefixed by dash (-). Some keyword options take the next item in the list as an argument. The *list_of_files* is a list of source, object, or library file names separated by blanks.

Options Syntax

Typical compiler option formats are:

TABLE 3-1 Options Syntax

Syntax Format	Example
<i>-flag</i>	-g
<i>-flagvalue</i>	-Dnostep
<i>-flag=value</i>	-xunroll=4
<i>-flag value</i>	-o outfile

The following typographical conventions are also used in this section of the manual when describing the individual options:

TABLE 3-2 Typographic Notations for Options

Notation	Meaning	Example: Text/Instance
[]	Square brackets contain arguments that are optional.	-O[n] -O4, -O
{ }	Curly brackets contain a set of choices for a required option.	-d{y n} -dy
	The “pipe” or “bar” symbol separates arguments, only <i>one</i> of which may be chosen.	-B{dynamic static} -Bstatic
:	The colon, like the comma, is sometimes used to separate arguments.	-Rdir[:dir] -R/local/libs:/U/a
...	The ellipsis indicates omission in a series.	-xinline= <i>fn</i> [,... <i>fn</i>] -xinline=alpha,dos

Brackets, pipe, and ellipsis are *meta characters* used in the descriptions of the options and are not part of the options themselves.

Some general guidelines for options are:

- `-lx` is the option to link with library `libx.a`. It is always safer to put `-lx` after the list of file names to insure the order libraries are searched.
- In general, processing of the compiler options is from left to right, allowing selective overriding of macro options (options that include other options).
 - The above rule does not apply to linker options.
 - However, some options, `-I`, `-L`, and `-R` for example, accumulate values rather than override previous values when repeated on the same command line.

Source files, object files, and libraries are compiled and linked in the order in which they appear on the command line.

Options Summaries

In this section, the compiler options are grouped by function to provide an easy reference. The details will be found on the pages in the following sections, as indicated.

Commonly Used Options

The Sun Fortran compilers have many features that are selectable by optional command-line parameters. The short list below of commonly used options is a good place to start.

TABLE 3-3 Commonly Used Options

Action	Option	Details
Debug—global program checking across routines for consistency of arguments, commons, and so on.	<code>-Xlist</code>	<code>"-Xlist[x]"</code> on page 3-73
Debug—produce additional symbol table information to enable the <code>dbx</code> and Sun WorkShop debugger.	<code>-g</code>	<code>"-g"</code> on page 3-34

TABLE 3-3 Commonly Used Options (continued)

Action	Option	Details
Performance—invoke the optimizer to produce faster running programs.	-O[n]	“-O[n]” on page 3-45
Performance—produce efficient compilation and run times for the native platform, using a set of predetermined options	-fast	“-fast” on page 3-27
Bind as dynamic (or static) any library listed later in the command: -Bdynamic, -Bstatic	-Bx	“-B{static dynamic}” on page 3-17
Library—Allow or disallow dynamic libraries for the entire executable: -dy, -dn	-dx	“-d{y n}” on page 3-23
Compile only—Suppress linking; make a .o file for each source file.	-c	“-c” on page 3-18
Output file—Name the executable output file <i>nm</i> instead of <i>a.out</i> .	-o nm	“-o nm” on page 3-47
Profile—Profile by <i>procedure</i> for <i>gprof</i> .	-pg	“-pg” on page 3-50

Debugging Options

The following options aid program debugging, including post-mortem, runtime, and static source code analysis. The most useful are listed first.

TABLE 3-4 Debugging Options

Action	Option	Details
Compile for use with the debugger.	-g	"-g" on page 3-34
Global source program checking for consistency.	-Xlist	"-Xlist[x]" on page 3-73
Check for subscripts out of range.	-C	"-C" on page 3-18
Undeclared variables—show a warning message.	-u	"-u" on page 3-59
Version ID—show ID along with name of each compiler pass.	-V	"-V" on page 3-59
Enable VMS Fortran 'D' debugging statements with -vax=debug.	-vax=v	"-vax=v" on page 3-60
Allow debugging by dbx without .o files.	-xs	"-xs" on page 3-79

Floating-Point Options

For the following floating-point options, the most significant are listed first.

TABLE 3-5 Floating-Point Options

Action	Option	Details
Turn on SPARC nonstandard floating-point (<i>SPARC</i>)	-fns	"-fns[={no yes}]" on page 3-29
Set IEEE rounding mode in effect at startup	-fround=r	"-fround=r" on page 3-31
Set IEEE trapping mode in effect at startup	-ftrap=t	"-ftrap=t" on page 3-33
Set floating-point optimization preferences	-fsimple=n	"-fsimple[=n]" on page 3-32
Se floating-point precision	-fprecision=p	"-fprecision=p" on page 3-31

Library Options

For the following library linking options, the most useful are listed first.

TABLE 3-6 Library Options

Action	Option	Details
Bind as dynamic or static next library listed on command.	-Bx	"-B{static dynamic}" on page 3-17
Allow or disallow dynamic libraries for executable.	-dx	"-d{y n}" on page 3-23
Build a dynamic shared library.	-G	"-G" on page 3-34

TABLE 3-6 Library Options *(continued)*

Action	Option	Details
Search for libraries in this directory first.	<code>-Ldir</code>	" <code>-Ldir</code> " on page 3-38
Link with library <code>libx</code> .	<code>-lx</code>	" <code>-lx</code> " on page 3-38
Link with the SunWorkShop Performance Library	<code>-xlic_lib=sunperf</code>	" <code>-xlic_lib=libs</code> " on page 3-72
Multithread safe libraries, low level threads.	<code>-mt</code>	" <code>-mt</code> " on page 3-41
No automatic libraries.	<code>-nolib</code>	" <code>-nolib</code> " on page 3-43
No inline templates.	<code>-nolibmil</code>	" <code>-nolibmil</code> " on page 3-44
No run path in executable.	<code>-norunpath</code>	" <code>-norunpath</code> " on page 3-44
Library—do not make library if relocations remain.	<code>-ztext</code>	" <code>-ztext</code> " on page 3-84

Licensing Options

The following options are for licensing.

TABLE 3-7 Licensing Options

Action	Option	Details
Do not queue the license request.	-noqueue	"-noqueue" on page 3-44
Show license server user IDs.	-xlicinfo	"-xlicinfo" on page 3-73

Performance Options

For the following performance options, those with the greatest significance are listed first.

TABLE 3-8 Performance Options

Action	Option	Details
Faster execution and compilation on native platform by using a set of options.	-fast	"-fast" on page 3-27
Optimize performance—set optimization level	-O[n]	"-O[n]" on page 3-45
Specify target hardware system.	-xtarget=t	"-xtarget=t" on page 3-81
Collect or use data for a profile to optimize (SPARC).	-xprofile=p	"-xprofile=p" on page 3-77
Align COMMON block data and enable multi-word load/stores. (SPARC)	-dalign	"-dalign" on page 3-20
Arithmetic—use simple arithmetic model.	-fsimple	"-fsimple[=n]" on page 3-32

TABLE 3-8 Performance Options (continued)

Action	Option	Details
Arithmetic—use SPARC non-standard floating point (SPARC).	-fns	"-fns[={no yes}]"
Use inline library	-libmil	"-libmil" on page 3-39
Traps—assume no memory-based traps (SPARC).	-xsafe=mem	"-xsafe=mem" on page 3-80
Unroll loops—advise optimizer to unroll loops <i>n</i> times.	-unroll= <i>n</i>	"-unroll= <i>n</i> " on page 3-59
Fast math—use special fast math routines (SPARC).	-xlibmopt	"-xlibmopt" on page 3-72
Architecture—specify target instruction set.	-xarch= <i>a</i>	"-xarch= <i>a</i> " on page 3-62
Chip—specify target processor.	-xchip= <i>c</i>	"-xchip= <i>c</i> " on page 3-66
Check data dependencies—analyze loops (SPARC).	-depend	"-depend" on page 3-22
Inline the listed user routines to optimize for speed.	-inline= <i>rlst</i>	"-inline= <i>fl[,...fn]</i> " on page 3-37
Optimize across all source files on command line	-xcrossfile	"-xcrossfile [= <i>n</i>]" on page 3-69
Do no optimizations that increase code size.	-xspace	"-xspace" on page 3-80

Parallelization Options

For the following parallelization options, those with the greatest impact in most situations are listed first. Parallelization options require a Sun WorkShop license. See the Fortran README files for details

TABLE 3-9 Parallelization Options (SPARC)

Action	Option	Details
Parallelize with <code>-autopar</code> <code>-explicitpar</code> <code>-depend</code> .	<code>-parallel</code>	" <code>-parallel</code> " on page 3-49
Parallelize explicitly marked loops.	<code>-explicitpar</code>	" <code>-explicitpar</code> " on page 3-24
Parallelize "reduction" loops.	<code>-reduction</code>	" <code>-reduction</code> " on page 3-54
Parallelize loops automatically.	<code>-autopar</code>	" <code>-autopar</code> " on page 3-16
Specify the style for MP directives (<code>cray</code> or <code>sun</code>).	<code>-mp=x</code>	" <code>-mp={sun cray}</code> " on page 3-41
Prepare loops for profiling parallelization.	<code>-zlp</code>	" <code>-zlp</code> " on page 3-83
List which loops are successfully parallelized.	<code>-loopinfo</code>	" <code>-loopinfo</code> " on page 3-39
Stack local variables to optimize for parallelization.	<code>-stackvar</code>	" <code>-stackvar</code> " on page 3-56
Show warnings about parallelization.	<code>-vpara</code>	" <code>-vpara</code> " on page 3-61

Profiling Options

The following options enable runtime profiling in the compiled program. Depending on the options, profiling is done at either the basic block, procedure, or loop level.

TABLE 3-10 Profiling Options

Action—Enable Profiling by:	Option	Details
Basic block (using <code>tcov</code> , old style).	<code>-a</code>	“ <code>-a</code> ” on page 3-14
Procedure (using <code>gprof</code>).	<code>-pg</code>	“ <code>-pg</code> ” on page 3-50
Procedure (using <code>prof</code>).	<code>-p</code>	“ <code>-p</code> ” on page 3-47
Loops with parallelization (<i>SPARC</i>).	<code>-loopinfo</code>	“ <code>-loopinfo</code> ” on page 3-39
Basic block (using <code>tcov</code> , new style).	<code>-xprofile=tcov</code>	“ <code>-xprofile=p</code> ” on page 3-77

Alignment Options

The following options are for specifying data alignment strategies.

TABLE 3-11 Alignment Options

Action	Option	Details
Align COMMON block data (<i>SPARC</i>).	<code>-f</code>	“ <code>-f</code> ” on page 3-26
Force 8-byte alignment on all data.	<code>-dbl_align_all=yes</code>	“ <code>-dbl_align_all=y</code> ” on page 3-22

TABLE 3-11 Alignment Options *(continued)*

Action	Option	Details
Align COMMON data and use multi-word load/store. (SPARC)	-dalign	"-dalign" on page 3-20
Allow misaligned data (SPARC).	-misalign	"-misalign" on page 3-40
Specify what VMS alignment features to use	-vax=v	"-vax=v" on page 3-60

Backward Compatibility and Legacy Options

The following options are provided for backward compatibility with earlier compiler releases, and certain Fortran legacy capabilities.

TABLE 3-12 Backward Compatibility Options

Action	Option	Details
Double default data sizes: use -xtypemap instead.	-r8 or -dbl	"-xtypemap=spec" on page 3-82
Allow assignment to constant arguments.	-copyargs	"-copyargs" on page 3-19
External names—make external names without underscores.	-ext_names=e	"-ext_names=e" on page 3-25
Nonstandard arithmetic—allow nonstandard arithmetic.	-fnonstd	"-fnonstd" on page 3-29
Optimize performance for the host system.	-native	"-native" on page 3-42

TABLE 3-12 Backward Compatibility Options *(continued)*

Action	Option	Details
Output—use old style list-directed output.	<code>-oldldo</code>	“ <code>-oldldo</code> ” on page 3-47
DO loops—use one trip DO loops.	<code>-onetrip</code>	“ <code>-onetrip</code> ” on page 3-47

Obsolescent Options

The following options are no longer functional in the current release of the f77 and f90 compilers. Their appearance on a compiler command does not cause an error, and no action is taken; they are ignored.

TABLE 3-13 Obsolescent Options

Original Intention	Option
Compile for Thread Analyzer	<code>-ztha</code>
Compile for f90browse (f90)	<code>-db</code>
Disable exception traps (f90)	<code>-fnonstop</code>

Options Reference

This section shows all f77 and f90 compiler command-line option flags, including various risks, restrictions, caveats, interactions, examples, and other details. Each description indicates platform availability of the option.

SPARC and x86 indicate availability of an option on SPARC and x86 platforms, respectively. (See the multiplatform release discussion in the Preface.) :

SPARC/x86	Indication Which Compiler The Option Is Available On That Platform
77	only available with <code>f77</code> on that platform
90	only available with <code>f90</code> on that platform
77/90	available with both <code>f77</code> and <code>f90</code> on that platform
P	not available on that platform

Note that `f90 2.0` is *only* available on SPARC systems.

Options that are not available for a compiler on a particular platform will still be accepted *silently* by the compiler. That is, the compiler will accept the option on the command-line on that platform without issuing a warning, but the option does nothing.

This options reference details each option flag.

-386

Compile for Intel 80386.

◆ **x86:77**

Generate code that exploits features available on Intel 80386 compatible processors.

-486

Compile for Intel 80486.

◆ **x86:77**

Generate code that exploits features available on Intel 80486 compatible processors. Code compiled with `-486` does run on 80386 hardware, but it may run slightly slower.

-a

Profile by basic block using `tcov`, old style.

◆ **SPARC: 77/90 x86:77**

This is the old style of basic block profiling for `tcov`. See `-xprofile=tcov` for information on the new style of profiling and the `tcov(1)` man page for more details. Also see the manual, *Analyzing Program Performance with Sun WorkShop*.

Insert code to count the times each basic block of statements is executed. This invokes a runtime recording mechanism that creates one `.d` file for every `.f` file at normal program termination. The `.d` file accumulates execution data for the corresponding source file. The `tcov(1)` utility can then be run on the source file(s) to generate statistics about the program. The summary output produced by `tcov` is written to `file.tcov` for each source file. `-pg` and `gprof` are complementary to `-a` and `tcov`.

If set at compile-time, the `TCOVDIR` environment variable specifies the directory where the `.d` and `.tcov` files are located. If this variable is not set, then the `.d` files remain in the same directory as the `.f` files.

The `-xprofile=tcov` and the `-a` options are compatible in a single executable. That is, you can link a program that contains some files which have been compiled with `-xprofile=tcov`, and others with `-a`. You cannot compile a single file with both options.

If you compile and link in separate steps, and you compile with `-a`, then be sure to link with `-a`. You can mix `-a` with `-On`; in some earlier versions `-a` overrode `-On`.

For details, see the chapter *Performance Profiling* in the *Fortran Programming Guide*.

`-ansi`

Identify many nonstandard extensions.

◆ **SPARC: 77/90 x86:77**

Warning messages are issued for any uses of non-standard Fortran 77 or Fortran 90 extensions in the source code.

`-arg=local`

Preserve actual arguments over ENTRY statements.

◆ **SPARC:77 x86:77**

When you compile a subprogram with alternate entry points with this option, `f77` uses *copy restore* to preserve the association of dummy and actual arguments. For example, the following program would require compilation with `-arg=local` to insure proper execution:

```

A = SETUP(ALPHA,BETA,GAMMA)
ZORK = FXGAMMA(GCONST)
...
FUNCTION SETUP(A1,A2,A3)
...
ENTRY FXGAMMA(F)
FXGAMMA = F*GAMMA
...
RETURN
END

```

Without this option, there is no guarantee that the correct values of the actual arguments from the `SETUP` call will be referenced when the routine is entered through `FXGAMMA`. Code that relies on `-arg=local` is nonstandard.

`-autopar`

Enable automatic loop parallelization.

◆ SPARC: 77/90

Finds and parallelizes appropriate loops for running in parallel on multiple processors. Analyzes loops for inter-iteration data dependencies and loop restructuring. If the optimization level is not specified `-O3` or higher, it will automatically be raised to `-O3`.

`-g` cancels `-autopar`. Debugging is facilitated by specifying `-g` without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the `dbx` documentation for details.

To improve performance, also specify the `-stackvar` option when using any of the parallelization options, including `-autopar`.

Avoid `-autopar` if the program already contains explicit calls to the `libthread` threads library. See note with `-mt` on “`-mt`” on page 3-41.

The `-autopar` option is not appropriate on a single-processor system, and the compiled code will generally run slower.

To run a parallelized program on a multiprocessor system, you must set the `PARALLEL` environment variable prior to execution. The default is 1. However, do not request more processors than are available.

If you use `-autopar` and compile and link in *one* step, the microtasking library and the threads-safe Fortran runtime library will automatically be linked. If you use `-autopar` and compile and link in *separate* steps, then you must also link with `-autopar` to insure linking the appropriate libraries.

Other parallelization options are `-parallel` and `-explicitpar`. Also, the `-reduction` option may be used with `-autopar`.

Refer to the *Fortran Programming Guide* for more information on parallelization.

`-B{static|dynamic}`

Prefer dynamic or require static library linking.

◆ SPARC: 77/90 x86:77

No space is allowed between `-B` and `dynamic` or `static`. The default, without `-B` specified, is `-Bdynamic`.

- `-Bdynamic`: Prefer *dynamic* linking (try for shared libraries).
- `-Bstatic`: Require *static* linking (no shared libraries).

Also note:

- If you specify `static`, but the linker finds only a dynamic library, then the library is not linked with a warning that the “library was not found.”
- If you specify `dynamic`, but the linker finds only a static version, then that library is linked, with no warning.

You can toggle `-Bstatic` and `-Bdynamic` on the command line. That is, you can link some libraries statically and some dynamically by specifying `-Bstatic` and `-Bdynamic` any number of times on the command line, as follows:

```
f77 prog.f -Bdynamic -lwells -Bstatic -lsurface
```

These are loader and linker options. Compiling and linking in separate steps with `-Bx` on the compile command will require it in the link step as well.

You cannot specify both `-Bdynamic` and `-dn` on the command line because `-dn` disables linking of dynamic libraries.

In a 64-bit Solaris environment, many system libraries are available only as shared dynamic libraries. These include `libm.so` and `libc.so` (`libm.a` and `libc.a` are not provided). This means that `-Bstatic` and `-dn` may cause linking errors in 64-bit Solaris environments. Applications must link with the dynamic libraries in these cases.

See the *Fortran Programming Guide* for more information on static and dynamic libraries.

-C

Check array references for out of range subscripts.

◆ **SPARC: 77/90 x86:77**

Subscripting arrays beyond their declared sizes may result in unexpected results, including segmentation faults. The **-C** option checks for possible array subscript violations in the source code and during execution.

Specifying **-C** may make the executable file larger.

If the **-C** option is used, array subscript violations are treated as an error. If an array subscript range violation is detected in the source code during compilation, it is treated as a compilation error.

If an array subscript violation can only be determined at runtime, the compiler generates range-checking code into the executable program. This may cause an increase in execution time. As a result, it is appropriate to enable full array subscript checking while developing and debugging a program, then recompiling the final production executable without subscript checking.

-C

Compile only; produce object `.o` files, but suppress linking.

◆ **SPARC: 77/90 x86:77**

Suppress linking. Compile a `.o` file for each source file. If only a single source file is being compiled, the **-o** option can be used to specify the name of the `.o` file written.

-cg89

Compile for generic SPARC architecture.

◆ **SPARC: 77/90**

This option is a macro for: `-xarch=v7 -xchip=old -xcache=64/32/1` which is equivalent to `-xtarget=ss2`.

-cg92

Compile for SPARC V8 architecture.

◆ **SPARC: 77/90**

This option is a macro for: `-xarch=v8 -xchip=super -xcache=16/32/4:1024/32/1` which is equivalent to `-xtarget=ss1000`.

`-copyargs`

Allow assignment to constant arguments.

◆ **SPARC:77 x86:77**

Allow a subprogram to change a dummy argument that is a constant. This option is provided only to allow legacy code to compile and execute without a runtime error.

- Without `-copyargs`, if you pass a constant argument to a subroutine, and then within the subroutine try to change that constant, the run aborts.
- With `-copyargs`, if you pass a constant argument to a subroutine, and then within the subroutine change that constant, the run does not necessarily abort.

Code that aborts unless compiled with `-copyargs` is, of course, not FORTRAN standard compliant. Also, such code is often unpredictable.

`-Dname [=def]`

Define symbol *name* for the preprocessor.

◆ **SPARC: 77/90 x86:77**

This option only applies to `.F` and `.F90` source files.

`-Dname=def` Define *name* to have value *def*

`-Dname` Define *name* to be 1

On the command line, this option will define *name* as if:

```
#define name[=def]
```

had appears in the source file. If no `=def` specified, the name *name* is defined as the value 1. The macro symbol *name* is passed on to the preprocessor `fpp` (or `cpp` — see the `-xpp` option) for expansion.

Following are the predefined values (these symbols have two leading underscores):

- The compiler version is predefined (in hex) in `__SUNPRO_F77` and `__SUNPRO_F90`

Example:

For FORTRAN 77 5.0, `__SUNPRO_F77=0x500` For Fortran 90 2.0,
`__SUNPRO_F90=0x200`

- The following values are predefined on appropriate systems:

```
__sparc, __unix, __sun, __i386, __SVR4,  
__SunOS_5_5_1, __SunOS_5_6, __SunOS_5_7
```

For instance, the value `__i386` is defined on systems compatible with the 80386 (including the 80486), and it is not defined on SPARC systems. You can use these values in such preprocessor conditionals as the following.

```
#ifdef __sparc
```

- From earlier releases, these values (with no underscores) are also predefined, but they may be deleted in a future release:

```
sparc, unix, sun, i386
```

- On SPARC V9 systems, the `__sparcv9` macro is also defined.

The compilers use the *fpp*(1) preprocessor by default. Like the C preprocessor *cpp*(1), *fpp* expands source code macros and enables conditional compilation of code. Unlike *cpp*, *fpp* understand Fortran syntax, and is preferred as a Fortran preprocessor. Use the `-xpp=cpp` flag to force the compiler to specifically use *cpp* rather than *fpp*.

-dalign

Align COMMON block data and generate faster multi-word load/stores.

◆ SPARC: 77/90

This flag changes the data layout in COMMON blocks (and EQUIVALENCE classes), and enables the compiler to generate faster multi-word load/stores for that data.

The data layout effect is that of the `-f` flag: double- and quad-precision data in COMMON blocks and EQUIVALENCE classes are laid out in memory along their “natural” alignment, which is on 8-byte boundaries (or on 16-byte boundaries for quad-precision when compiling for 64-bit environments with `-xarch=v9` or `v9a`). The default alignment of data in COMMON blocks is on 4-byte boundaries. The compiler is also allowed to assume natural alignment and generate faster multi-word load/stores to reference the data.

Note - `-dalign` may result in nonstandard alignment of data, which could cause problems with variables in EQUIVALENCE or COMMON and may render the program non-portable if `-dalign` is required.

Using both `-dbl` and `-dalign` also causes default INTEGER variables to be 8-byte aligned and 64-bits. Also: “`-xtypemap=real:x,double:y,integer:64`”

If you compile one subprogram with `-dalign`, compile all subprograms of the program with `-dalign`. This option is included in the `-fast` option.

`-dbl`

Double the default size for REAL, INTEGER, DOUBLE, and COMPLEX.

◆ SPARC:77 x86:77

Note - This option, and `-r8`, are now considered obsolete and may be removed in future releases. Use the more general `-xtypemap` option instead.

`-dbl` promotes the default byte size for REAL, INTEGER, DOUBLE, and COMPLEX variables declared *without an explicit byte size* as follows:

TABLE 3-14 Default Data Sizes and `-dbl` (Bytes)

Without <code>-dbl</code> option		With <code>-dbl</code> option	
Data Type	default	SPARC	x86
INTEGER	4	8	8
REAL	4	8	8
DOUBLE	8	16	8

This option applies to variables, parameters, constants, and functions.

Also, LOGICAL is treated as INTEGER, COMPLEX as two REALS, and DOUBLE COMPLEX as two DOUBLES.

Compare `-dbl` with `-r8`: `-dbl` and `-r8` can be expressed in terms of the more general `-xtypemap=` option:

■ On SPARC:

`-dbl equals: -xtypemap=real:64,double:128,integer:64` `-r8 equals: -xtypemap=real:64,double:1`

These options promote default DOUBLE PRECISION data to QUAD PRECISION (128 bits). This may be unwanted and may cause performance degradation.

`-xtypemap=real:64,double:64,integer:64` might be more appropriate than `-dbl` in these cases.

- On x86:

`-dbl equals: -xtypemap=real:64,double:64,integer:64` `-r8 equals: -xtypemap=real:64,double:64,integer:64`

- For all of the floating point data types, `-dbl` works the same as `-r8`; using both `-r8` and `-dbl` produces the same results as using only `-dbl`.
- For INTEGER and LOGICAL data types, `-dbl` is different from `-r8`:
 - `-dbl` allocates 8 bytes, and does 8-byte arithmetic
 - `-r8` allocates 8 bytes, and does only 4-byte arithmetic (“mixed”)

In general, if you compile *one* subprogram with `-dbl`, then be sure to compile *all* subprograms of that program with `-dbl`. This is particularly important with programs communicating through files with unformatted I/O — if one program is compiled with `-dbl`, then the other program must similarly be compiled. Be also aware that this option alters the default data size of function names, including calls to library functions, unless the function name is typed explicitly with a data size.

`-dbl_align_all=y`

Force alignment of all data on 8-byte boundaries

- ◆ SPARC:77 x86:77

`y` is either `yes` or `no`. If `y` is `yes`, all variables will be aligned on 8-byte boundaries. Default is `-dbl_align_all=no`.

When compiling for 64-bit environments with `-xarch=v9` or `v9a`, this flag will align quad-precision data on 16-byte boundaries.

This flag does not alter the layout of data in COMMON blocks or user-defined structures.

On SPARC, use with `-dalign` to enable added efficiency with multi-word load/stores.

If used, all routines must be compiled with this flag.

`-depend`

Analyze loops for data dependencies.

- ◆ SPARC: 77/90

Analyze loops for inter-iteration data dependencies and do loop restructuring. This option will raise the optimization level to O3 if no optimization level is specified, or if it is specified less than O3. `-depend` is also included with `-fast`, `-autopar` and `-parallel`. (See the *Fortran Programming Guide*.)

`-g` cancels `-depend`.

`-dryrun`

Show commands built by driver, but do not compile.

◆ **SPARC: 77/90 x86:77**

Useful when debugging, this option displays the commands it will run to perform the compilation.

`-d{y|n}`

Allow or disallow *dynamic* libraries for the entire executable

◆ **SPARC: 77/90 x86:77**

- `-dy`: Yes, allow *dynamically* bound libraries (*allow* shared libraries).
- `-dn`: No, do *not* allow dynamically bound libraries (*no* shared libraries).

The default, if not specified, is `-dy`.

Unlike `-Bx`, this option applies to the *whole* executable and need appear only once on the command line.

`-dy|dn` are loader and linker options. If you compile and link in separate steps with these options, then you need the same option in the link step.

In a 64-bit Solaris environment, many system libraries are not available only as shared dynamic libraries. These include `libm.so` and `libc.so` (`libm.a` and `libc.a` are not provided). This means that `-dn` and `-Bstatic` may cause linking errors in 64-bit Solaris environments. Applications must link with the dynamic libraries in these cases.

`-e`

Accept extended length input source line.

◆ **SPARC: 77/90 x86:77**

Accept source lines up to 132 characters long. The compiler pads on the right with trailing blanks to column 132. If you use continuation lines while compiling with `-e`, then do not split character constants across lines, otherwise, unnecessary blanks may be inserted in the constants.

`-eroff=taglist`

Suppress warning messages listed by tag name.

◆ **SPARC:77 x86:77**

Suppress displaying the warning messages specified in the comma-separated list of tag names *taglist*. If *taglist* consists of `%none`, no warnings are suppressed. If *taglist* consists of `%all`, all warnings are suppressed (this is equivalent to the `-w` option.)

Example:

```
f77 -eroff=WDECL_LOCAL_NOTUSED ink.f
```

Use the `-errtags` option to see the tag names associated with warning messages.

`-errtags`

Display the message tag with each warning message.

◆ **SPARC:77 x86:77**

With this option, the compiler's internal error tag name will appear along with warning messages. The default is not to display the tag.

```
demo% f77 -errtags ink.f
ink.f:
  MAIN:
```

```
"ink.f", line 11: Warning: local variable "i" never used (WDECL_LOCAL_NOTUSED) <- The warning message's tag na
```

`-explicitpar`

Parallelize loops explicitly marked by directives.

◆ **SPARC: 77/90**

The compiler will generate parallel code even if there are data dependencies in the DO loop that would cause the loop to generate incorrect results when run in parallel. With explicit parallelization, it is the user's responsibility to correctly analyze loops for data dependency problems before marking them with parallelization directives.

This option turns on explicit parallelization. DO loops immediately preceded by DOALL directives will have threaded, parallel code compiled for them. Parallelization is only appropriate on multiprocessor systems. This option should not be used to compile programs that already do their own multithreading with calls to the `libthread` library.

If you use `-explicitpar` and compile and link in *one* step, then linking automatically includes the microtasking library and the threads-safe FORTRAN runtime library. If you use `-explicitpar` and compile and link in *separate* steps, then you must also *link* with `-explicitpar`.

To improve performance, also specify the `-stackvar` option when using any of the parallelization options, including `-explicitpar`.

If the optimization level is not `-O3` or higher, it is raised to `-O3` automatically.

For details, see the *Parallelization* chapter in the *Fortran Programming Guide*.

`-g` cancels `-explicitpar`. Debugging is facilitated by specifying `-g` without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the `dbx` documentation for details.

`-ext_names=e`

Create external names with or without trailing underscores.

◆ **SPARC:77 x86:77**

`e` must be either `plain` or `underscores`. The default is `underscores`.

`-ext_names=plain`: Do not add trailing underscore.

`-ext_names=underscores`: Add trailing underscore.

An external name is a name of a subroutine, function, block data subprogram, or labeled common. This option affects both the name of the routine's entry point and the name used in calls to it. This option may be used to allow Fortran 77 routines to call and be called by other language routines.

-F

Invoke the source file preprocessor, but do not compile.

◆ SPARC: 77/90 x86:77

Apply the `fpp` preprocessor to `.F` files (and `.F90` files with `f90`) and write the processed result on a file with the same name but with suffix changed to `.f` (or `.f90`), but do not compile.

Example:

```
f77 -F source.F
```

writes the processed source file to `source.f`

`fpp` is the default preprocessor for Fortran. The C preprocessor, `cpp`, can be selected instead by specifying `-xpp=cpp`.

-f

Align data in COMMON blocks.

◆ SPARC: 77/90

Align double- and quad-precision data in COMMON blocks.

This flag changes the data layout in COMMON blocks (and EQUIVALENCE classes): double- and quad-precision data in COMMON blocks and EQUIVALENCE classes are laid out in memory along their “natural” alignment, which is on 8-byte boundaries (or on 16-byte boundaries for quad-precision when compiling for 64-bit environments with `-xarch=v9` or `v9a`). The default alignment of data in COMMON blocks is on 4-byte boundaries.

Note - `-f` may result in nonstandard alignment of data, which could cause problems with variables in EQUIVALENCE or COMMON and may render the program non-portable if `-f` is required.

Using `-dbl` with `-f` aligns all 64-bit integer data on 8-byte boundaries as well.

Compiling *any* part of a program with `-f` requires compiling *all* subprograms of that program with `-f`.

By itself, this option does not enable the compiler to generate faster multi-word fetch/store instructions on double and quad precision data. The `-dalign` option does this and invokes `-f` as well. Use of `-dalign` is preferred over the older `-f`. See `-dalign`, “`-dalign`” on page 3-20. Because `-dalign` is part of the `-fast` option, so is `-f`.

-fast

Optimize for speed of execution using a selection of options.

◆ SPARC: 77/90 x86:77

Select options that optimize for speed of execution without excessive compilation time. This option provides close-to-the-maximum performance for many applications.

If you compile and link in separate steps, and you compile with `-fast`, then be sure to link with `-fast`.

Note - This option is defined as a particular selection of other options that is subject to change from one release to another, and between compilers. Also, some of the options selected by `-fast` may not be available on some platforms.

TABLE 3-15 `-fast` selections across platforms

SPARC	x86
<code>-dalign</code>	—
<code>-depend</code>	—
<code>-fns</code>	<code>-fns</code>
<code>-fsimple=1</code>	—
<code>-ftrap=%none</code>	<code>-ftrap=%none</code>
<code>-libmil (f77) -f (f90)</code>	<code>-libmil</code>
<code>-native</code>	<code>-native</code>
—	<code>-nofstore</code>
<code>-O4</code>	<code>-O4</code>
<code>-xlibmopt</code>	<code>-xlibmopt</code>

Details about the options selected by `-fast`:

- The `-native` hardware target. If the program is intended to run on a different target than the compilation machine, follow the `-fast` with a code-generator option. For example: `f77 -fast xtarget=ultra ...`
- The `-O4` optimization level option.

- The `-depend` option (*SPARC only*).
- The `-libmil` option for system-supplied inline expansion templates. For C functions that depend on exception handling, follow `-fast` by `-nolibmil: -fast -nolibmil`. With `-libmil`, exceptions cannot be detected with `errno` or `matherr(3m)`.
- The `-fsimple=1` option for a simple floating-point model. `-fsimple` is unsuitable if strict IEEE 754 standards compliance is required.
- The `-dalign` option to generate double loads and stores (*SPARC only*). Using this option may generate nonstandard Fortran data alignment.
- The `-xlibmopt` option (*SPARC only*)
- `-nofstore` (*x86 only*)
- `-fns -ftrap=%none` to turn off all trapping.

Note - With `f90`, the `-f` option is substituted for `-libmil` on SPARC.

It is possible to add or subtract from this list by following the `-fast` option with other options, as in:

```
f77 -fast -fsimple=2 -xnolibmopt ...
```

which overrides the `-fsimple=1` option and disables the `-xlibmopt` selected by `-fast`.

`-fixed`

Specify fixed-format Fortran 90 source input files.

◆ SPARC:90

All source files on the command-line will be interpreted as `f77` fixed format regardless of filename extension. Normally, `f90` interprets only `.f` files as fixed format, `.f90` as free format.

`-flags`

Synonym for `-help`.

◆ SPARC: 77/90 x86:77

`-fnonstd`

Initialize floating-point hardware to non-standard preferences.

◆ SPARC: 77/90 x86:77

This option is a synonym for the combination of the following option flags:

- SPARC: `-fns -ftrap=common`
- x86: `-ftrap=common`

Specifying `-fnonstd` is approximately equivalent to the following two calls at the beginning of a Fortran main program.

```
i=ieee_handler("set", "common", SIGFPE_ABORT)
call nonstandard_arithmetic()
```

The `nonstandard_arithmetic()` routine replaces the obsolete `abrupt_underflow()` routine of earlier releases.

To be effective, the main program must be compiled with this option.

Using this option initializes the floating-point hardware to:

- Abort (trap) on floating-point exceptions.
- SPARC: Flush underflow results to zero if it will improve speed, rather than produce a subnormal number as the IEEE standard requires.

See `-fns` for more information about gradual underflow and subnormal numbers.

The `-fnonstd` option allows hardware traps to be enabled for floating-point overflow, division by zero, and invalid operation exceptions. These are converted into SIGFPE signals, and if the program has no SIGFPE handler, it terminates with a dump of memory.

For more information, see the `ieee_handler(3m)` and `ieee_functions(3m)` man pages, the *Numerical Computation Guide*, and the *Fortran Programming Guide*.

`-fns [= {no | yes}]`

Select the SPARC nonstandard floating-point mode.

◆ SPARC: 77/90

The default is the SPARC standard floating-point mode (`-fns=no`). (See the *Floating-Point Arithmetic* chapter of the *Fortran Programming Guide*.)

Optional use of `=yes` or `=no` provides a way of toggling the `-fns` flag following some other macro flag that includes it, such as `-fast`. `-fns` is the same as `-fns=yes`.

This option flag enables nonstandard floating-point mode when the program begins execution. On some SPARC systems, specifying nonstandard floating-point mode disables “gradual underflow”, causing tiny results to be flushed to zero rather than producing subnormal numbers. It also causes subnormal operands to be silently replaced by zero. On those SPARC systems that do not support gradual underflow and subnormal numbers in hardware, use of this option can significantly improve the performance of some programs.

Where x does not cause total underflow, x is a *subnormal number* if and only if $|x|$ is in one of the ranges indicated:

TABLE 3-16 Subnormal REAL and DOUBLE

Data Type	Range
REAL	$0.0 < x < 1.17549435e-38$
DOUBLE PRECISION	$0.0 < x < 2.22507385072014e-308$

See the *Numerical Computation Guide* for details on subnormal numbers, and the *Fortran Programming Guide* chapter *Floating-Point Arithmetic* for more information about this and similar options. (Some arithmeticians use the term *denormalized number* for *subnormal number*.)

The standard initialization of floating-point preferences is the default:

- IEEE 754 floating-point arithmetic is *nonstop* (do not abort on exception).
- Underflows are gradual.

To be effective, the main program must be compiled with this option.

`-fpover [= {yes | no}]`

Detect floating-point overflow in formatted input.

◆ **SPARC: 77/90 x86:77**

With `-fpover=yes` specified, the I/O library will detect runtime floating-point overflows in formatted input and return an error condition (1031). The default is no such overflow detection (`-fpover=no`). `-fpover` is equivalent to `-fpover=yes`.

`-fprecision=p`

Initialize floating-point precision mode on x86.

◆ x86:77

p is either `single`, `double`, or `extended`.

Initialize the floating-point hardware precision mode on x86 platforms to `single`, `double`, or `extended`. Compile the main program with this option. (See the *Floating-Point* chapter of the *Fortran Programming Guide*.)

The default, when `-fprecision` is not specified, is `-fprecision=extended`.

`-free`

Specify free-format source input files.

◆ SPARC:90

All source files on the command-line will be interpreted as `f90` free format regardless of filename extension. Normally, `f90` interprets `.f` files as fixed format, `.f90` as free format.

`-fround=r`

Set the IEEE rounding mode in effect at startup.

◆ SPARC: 77/90 x86:77

r must be one of: `nearest`, `tozero`, `negative`, `positive`.

The default is `-fround=nearest`.

To be effective, compile the main program with this option.

This option sets the IEEE 754 rounding mode that:

- Can be used by the compiler in evaluating constant expressions.
- Is established at runtime during the program initialization.

When *r* is `tozero`, `negative`, or `positive`, the option sets the rounding direction to *round-to-zero*, *round-to-negative-infinity*, or *round-to-positive-infinity*, respectively, when the program begins execution. When `-fround` is not specified, `-fround=nearest` is used as the default and the rounding direction is

round-to-nearest. The meanings are the same as those for the `ieee_flags` function. (See the *Floating-Point Arithmetic* chapter of the *Fortran Programming Guide*.)

`-fsimple [=n]`

Select floating-point optimization preferences.

◆ SPARC: 77/90 x86:77

Allow the optimizer to make simplifying assumptions concerning floating-point arithmetic. (See the *Floating-Point Arithmetic* chapter of the *Fortran Programming Guide*.)

For consistent results, compile all units of a program with the same `-fsimple` option.

If *n* is present, it must be 0, 1, or 2. The defaults are:

- Without the `-fsimple` flag, the compiler defaults to `-fsimple=0`
- With `-fsimple` alone, the compiler defaults to `-fsimple=1`

The different floating-point simplification levels are:

`-fsimple=0`

Permit no simplifying assumptions. Preserve strict IEEE 754 conformance.

`-fsimple=1`

Allow conservative simplifications. The resulting code does not strictly conform to IEEE 754, but numeric results of most programs are unchanged.

With `-fsimple=1`, the optimizer can assume the following:

- IEEE 754 default rounding/trapping modes do not change after process initialization.
- Computations producing no visible result other than potential floating point exceptions may be deleted.
- Computations with Infinity or NaNs (“Not a Number”) as operands need not propagate NaNs to their results; e.g., $x*0$ may be replaced by 0.
- Computations do not depend on sign of zero.

With `-fsimple=1`, the optimizer is *not* allowed to optimize completely without regard to roundoff or exceptions. In particular, a floating-point computation cannot be replaced by one that produces different results with rounding modes held constant at run time. `-fast` includes `-fsimple=1`.

`-fsimple=2`

Permit aggressive floating point optimizations that may cause many programs to produce different numeric results due to changes in rounding.

For example, `-fsimple=2` permits the optimizer to attempt to replace repeated computations of x/y with $x*z$, where $z=1/y$ is computed once and saved in a temporary, eliminating the costly divide operation.

Even with `-fsimple=2`, the optimizer still is not permitted to introduce a floating point exception in a program that otherwise produces none.

`-fstore`

Force precision of floating-point expressions.

◆ **x86:77**

Use the precision of destination variable to determine the precision of the right-hand-side expression on assignment statements on x86 platforms. If not specified, the default is `-fstore`.

The `-fast` option sets `-nofstore` to disable this option. Follow the `-fast` flag on the command line with `-fstore` to turn this option back on.

`-ftrap=t`

Set floating-point trapping mode in effect at startup.

◆ **SPARC: 77/90 x86:77**

t is a comma-separated list that consists of one or more of the following:

`%all`, `%none`, `common`, `[no%]invalid`, `[no%]overflow`, `[no%]underflow`,
`[no%]division`, `[no%]inexact`.

Where the `%` is shown, it is a required character.

The `f77` default is `-ftrap=%none`. The `f90` default is `-ftrap=common`.

This option sets the IEEE 754 trapping modes that are established at program initialization. Processing is left-to-right. The common exceptions, by definition, are invalid, division by zero, and overflow. For example: `-ftrap=overflow`.

Example: `-ftrap=%all,no%inexact` means set all traps, except `inexact`.

The meanings for `-ftrap=t` are the same as for `ieee_flags()`, except that:

- `%all` turns on all the trapping modes.
- `%none`, the default, turns off all trapping modes.
- A `no%` prefix turns off that specific trapping mode.

To be effective, compile the main program with this option.

For further information, see the *Floating-Point Arithmetic* chapter in the *Fortran Programming Guide*.

-G

Build a dynamic shared library instead of an executable file.

◆ **SPARC: 77/90 x86:77**

Direct the linker to build a *shared dynamic* library. Without **-G**, the linker builds an executable file. With **-G**, it builds a dynamic library. Use **-o** with **-G** to specify the name of the file to be written. See the *Fortran Programming Guide* chapter *Libraries* for details.

-g

Compile for debugging.

◆ **SPARC: 77/90 x86:77**

Produce additional symbol table information for the debugging with *dbx(1)* and the Sun WorkShop debugging utility.

Although a some debugging is possible without specifying **-g**, the full capabilities of *dbx* and *debugger* are only available to those compilation units compiled with **-g**.

Some capabilities of other options specified along with **-g** may be limited. The **-g** option suppresses the automatic inlining usually obtained with **-O4**, but it does not suppress **-On** optimizations.

-g cancels any parallelization option (**-autopar**, **-explicitpar**, **-parallel**) as well as **-depend** and **-reduction**. Debugging is facilitated by specifying **-g** without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the *dbx* documentation for details.

For *x86*: **-g** is ignored when specified with a **-On** option or **-fast**.

For *SPARC*: The **-g** option makes **-xildon** the default incremental linker option (see “**-xildon**” on page 3-71). That is, with **-g**, the compiler default behavior is to automatically invoke *ild* in place of *ld*, unless the **-G** option is present, or any source file is named on the command line.

-hnm

Specify the name of the generated dynamic shared library.

◆ **SPARC: 77/90 x86:77**

This option is passed on to the linker. For details, see the *Solaris Linker and Libraries Guide*, and the *Fortran Programming Guide* chapter *Libraries*.

The `-hnm` option records the name `nm` to the shared dynamic library being created as the internal name of the library. A space between `-h` and `nm` is optional (except if the library name is `elp`, for which the space will be needed). In general, `nm` must be the same as what follows the `-o`. Use of this option is meaningless without also specifying `-G`.

Without the `-hnm` option, no internal name is recorded in the library file.

If the library has an internal name, whenever an executable program referencing the library is run the runtime linker will search for a library with the same internal name in any path the linker is searching. With an internal name specified, searching for the library at runtime linking is more flexible. This option can also be used to specify *versions* of shared libraries.

If there is no internal name of a shared library, then the linker uses a specific path for the shared library file instead.

-help

Display a summary list of compiler options.

◆ **SPARC: 77/90 x86:77**

Displays a list of option summaries and indicates how to send feedback comments to Sun. See also `-xhelp=h` on “`-xhelp=h`” on page 3-70.

-I dir

Add `dir` to the INCLUDE file search path.

◆ **SPARC: 77/90 x86:77**

Insert the directory `dir` at the start of the INCLUDE file search path. No space is allowed between `-I` and `dir`. Invalid directories are ignored with no warning message.

The *include file search path* is the list of directories searched for INCLUDE files—file names appearing on preprocessor #include directives, or Fortran INCLUDE statements.

Example: Search for INCLUDE files in /usr/app/include:

```
demo% f77 -I/usr/app/include growth.F
```

Multiple `-I` options may appear on the command line. Each adds to the top of the search path list (first path searched).

The search order for relative path on INCLUDE or #include is:

1. The directory that contains the source file
2. The directories that are named in the `-I` options
3. The directories in the default list

The default list for `-I` depends on the installation directory for the compiler. In a standard install, compiler software packages reside in the `/opt` directory; however, systems administrators may decide to install packages in other locations. If an environment variable, `INSTALL_HOME` say, points at the installation path (e.g. `/opt`, or `/some/place`), the default search paths for INCLUDE files are:

- for f77: `$INSTALL_HOME/SUNWspro/SC5.0/include/f77 /usr/include`
- for f90: `$INSTALL_HOME/SUNWspro/SC5.0/include/f90 /usr/include`

`-i 2`

Set the default integer size to two bytes.

◆ **SPARC:77 x86:77**

Set the default size to 2 bytes for integer and logical constants and variables declared without an explicit size. (`INTEGER*n Y` still declares `Y` to be `n` bytes regardless of the `-i2`.) This option may degrade performance. It is generally recommended to declare specific variables `INTEGER*2` rather than use `-i2`.

`-i 4`

Set the default integer size to four bytes.

◆ **SPARC:77 x86:77**

Set the default size to 4 bytes for integer and logical constants and variables declared without an explicit size. (`INTEGER*n Y` still declares `Y` to be `n` bytes regardless of the `-i4`).

Although 4 bytes *is* the default size for `INTEGER` and `LOGICAL`, this option can be used for overriding settings made by options like `-dbl` and `-r8`, which set these defaults to 8:

```
demo% f77 -dbl -i4 *.f
Command line warning: --i4 overrides integer part of --dbl
...
```

`-inline=f1[,...fn]`

Inline specified routines.

◆ SPARC: 77/90 x86:77

Request that the optimizer inline the user-written routines named in the `f1,...,fn` list. Inlining is an optimization technique whereby the compiler effectively replaces a subprogram reference such as a `CALL` or function call with the actual subprogram code itself. Inlining often provides the optimizer more opportunities to produce efficient code.

The list is a comma-separated list of functions and subroutines.

Example: Inline the routines `xbar`, `zbar`, `vpoint`:

```
demo% f77 -O3 -inline=xbar,zbar,vpoint *.f
```

Following are the restrictions; no warnings are issued:

- SPARC: Optimization must be `-O3` or greater.
- The source for the routine must be in the file being compiled, unless `-xcrossfile` is also specified.
- The compiler determines if actual inlining is profitable and safe.

With `-O4`, the compilers normally try to inline all appropriate user-written subroutines and functions. Adding `-inline` with `-O4` actually degrades performance by restricting the optimizer's inlining to only those routines in the list.

`-Kpic`

Synonym for `-pic`.

◆ SPARC: 77/90 x86:77

-KPIC

Synonym for -PIC.

◆ SPARC: 77/90 x86:77

-Ldir

Add *dir* to list of directories to search for libraries.

◆ SPARC: x86:77

Add *dir* at the *start* of the list of object-library search directories.

A space between -L and *dir* is optional. This option is passed to the linker.

See also -lx on “-lx” on page 3-38.

While building the executable file, *ld*(1) searches *dir* for archive libraries (.a files) and shared libraries (.so files). *ld* searches *dir* before searching the default directories.

(See the *Fortran Programming Guide* chapter *Libraries* for information on library search order.) For the relative order between LD_LIBRARY_PATH and -Ldir, see *ld*(1).

Example: Use -Ldir to specify library search directories:

```
demo% f77 -Ldir1 -Ldir2 any.f
```

Note - Specifying /usr/lib or /usr/ccs/lib with -Ldir may prevent linking the unbundled libm. These directories are searched by default.

-lx

Add library libx.a to linker's list of search libraries.

◆ SPARC: 77/90 x86:77

Pass -lx to the linker to specify additional libraries for *ld* to search for unresolved references. *ld* links with object library libx. If shared library libx.so is available (and -Bstatic or -dn are not specified), *ld* uses it, otherwise, *ld* uses static

library `libx.a`. If it uses a shared library, the name is built in to `a.out`. No space is allowed between `-l` and `x` character strings.

Example: Link with the library `libv77`:

```
demo% f77 any.f -lv77
```

Use `-lx` again to link with more libraries.

Example: Link with the libraries `liby` and `libz`:

```
demo% f77 any.f -ly -lz
```

See also the *Libraries* chapter in the *Fortran Programming Guide* for information on library search paths and search order.

`-libmil`

Inline selected `libm` library routines for optimization.

◆ **SPARC: 77/90 x86:77**

There are inline templates for some of the `libm` library routines. This option selects those inline templates that produce the fastest executable for the floating-point options and platform currently being used. The routines include the following:

`d_infinity`, `d_max_normal`, `d_max_subnormal`, `d_min_normal`, `d_min_subnormal`, `d_quiet_nan`,

This list of routines may change with subsequent compiler releases. For more information, see the man pages `libm_single(3F)` and `libm_double(3F)`

`-loopinfo`

Show parallelization results.

◆ **SPARC: 77/90**

Show which loops parallelized and which did not with the `-parallel`, `-autopar`, or `-explicitpar` options. (Option `-loopinfo` must appear with one of these parallelization options.)

`-loopinfo` generates a list of messages on standard error:

```

demo% f77 -o shalow -fast -parallel -loopinfo shalow.f
shalow.f:
  MAIN shalow:
    inital:
    calcl:
  ... etc
"shalow.f", line 78: not parallelized, call may be unsafe
"shalow.f", line 172: PARALLELIZED
"shalow.f", line 173: not parallelized, not profitable
"shalow.f", line 181: PARALLELIZED, fused
"shalow.f", line 182: not parallelized, not profitable
"shalow.f", line 226: PARALLELIZED, and serial version generated
"shalow.f", line 227: not parallelized, not profitable
... etc

```

Use the *error(1)* utility with *f77* compilations to merge this list with the source file to produce an annotated source listing with each loop tagged as parallelized or not.

Example: *-loopinfo*, in *sh*, pass standard error to the *error* utility:

```

demo$ f77 -autopar -loopinfo any.f 2>&1 | error
options

```

Be aware that *error* rewrites the input source file. For details on *error*, see the *error* man page and the *Fortran Programming Guide* chapter on debugging.

-Mdir

Add *dir* to directories searched for Fortran 90 modules.

◆ SPARC:90

Add *dir* to the list of directories to be searched for module files. No space appears between the *-M* and *dir*.

The directories listed with *-M* are searched after the current directory. Compiling a source file containing a module generates a *.mod* module file for each *MODULE* encountered. See Appendix C, "Module Files" on page C-22 for more information about modules in Fortran 90.

-misalign

Allow misaligned data.

◆ SPARC:77

The `-misalign` option permits misaligned data in memory that would otherwise produce an error. Particular uses of `COMMON` and `EQUIVALENCE` statements may cause data to be misaligned (with a compiler diagnostic). With `-misalign`, the compiler will allow intentional misalignment and will not add padding in `COMMON` blocks to insure proper data alignment. However, this seriously degrades performance; recoding to eliminate the cause of data misalignment is a better alternative.

If used, all routines in a program must be compiled with this option. If you compile and link in separate steps, compiling with the `-misalign` option requires the option on the link step as well.

`-mp = { sun | cray }`

Select the style for parallelization directives.

◆ **SPARC: 77/90**

The default without specifying `-mp` is `sun`. Do not combine use in a single application.

`-mp=sun`: Accept only the Sun-style directives: `C$PAR` or `!$PAR` prefix.

`-mp=cray`: Accept only the Cray-style directives: `CMIC$` or `!MIC$` prefix.

See the *Fortran Programming Guide* chapter on *Parallelization* for details.

`-mt`

Require multithread-safe libraries.

◆ **SPARC: 77/90 x86:77**

Require linking to multithread-safe libraries. If you do your own low-level thread management (e.g. calls to the `libthread` library), compiling with `-mt` prevents conflicts.

Use `-mt` if you mix C and Fortran, and you manage multithread C coding using the `libthread` primitives. Before you use your own multi-threaded coding, read the *Solaris Multithreaded Programming Guide*.

The equivalent of `-mt` is included automatically with the `-autopar`, `-explicitpar`, or `-parallel` options.

Note the following:

- A function subprogram that does I/O should not itself be referenced as part of an I/O statement. Such *recursive* I/O may cause the program to deadlock with `-mt`.

- In general, do *not* compile your own multi-threaded coding with `-autopar`, `-explicitpar`, or `-parallel`. The compiler's generated calls to the threads library primitives any the programs own calls may conflict, causing unexpected results.
- On a single-processor system, performance may be degraded with the `-mt` option.

`-native`

Optimize performance for the host system.

- ◆ **SPARC: 77/90 x86:77**

This option is a synonym for `-xtarget=native`. This is one of the options included in the expansion of the `-fast` option .

`-noautopar`

Disable automatic parallelization.

- ◆ **SPARC: 77/90**

Disables automatic parallelization invoked by `-autopar` earlier on the command line.

`-nodepend`

Cancel `-depend` in command line.

- ◆ **SPARC: 77/90**

Cancel any `-depend` appearing earlier on the command line.

`-noexplicitpar`

Disable explicit parallelization.

- ◆ **SPARC: 77/90**

Disables explicit parallelization invoked by `-explicitpar` earlier on the command line.

`-nofstore`

Disable forcing precision of expression.

◆ **x86:77**

Disables forcing precision of expressions in assignment statements invoked by `-fstore` earlier on the command line (*x86 only*). `-nofstore` is invoked if `-fast` is specified.

`-nolib`

Disable linking with system libraries.

◆ **SPARC: 77/90 x86:77**

Do *not* automatically link with *any* system or language library; that is do *not* pass any default `-lx` options on to `ld`. The normal behavior is to link system libraries into the executables automatically, without the user specifying them on the command line.

The `-nolib` option makes it easier to link one of these libraries statically. The system and language libraries are required for final execution. It is your responsibility to link them in manually. This option provides you with complete control.

For example, consider a program linked dynamically with `libF77` that fails on a remote system because has no `libF77`. With this option you can link the library into your program statically.

Link `libF77` statically and link `libc` dynamically with `f77`:

```
demo% f77 -nolib any.f -Bstatic -lF77 -Bdynamic -lm -lc
```

Link `libm` statically and `libc` dynamically with `f90`:

```
demo% f90 -nolib any.f90 -Bstatic -lm -Bdynamic -lc
```

The order for the `-lx` options is important. Follow the order shown in the examples.

`-nolibmil`

Cancel `-libmil` on command line.

◆ **SPARC: 77/90 x86:77**

Use this option *after* the `-fast` option to disable inlining of `libm` math routines:

```
demo% f77 -fast -nolibmil
```

`-noqueue`

Disable license queueing.

◆ **SPARC: 77/90 x86:77**

With this option, if no software license is available to run the compiler, it returns without queueing your request and without compiling. A nonzero environment status is returned for testing in `make` files.

`-noreduction`

Cancel `-reduction` on command line.

◆ **SPARC: 77/90**

`-reduction` is used with other parallelization options. This option cancels `-reduction`.

`-norunpath`

Do not build a runtime shared library search path into the executable.

◆ **SPARC: 77/90 x86:77**

The compiler normally builds into an executable a path that tells the runtime linker where to find the shared libraries it will need. The path is installation dependent. The `-norunpath` option prevents that path from being built in to the executable.

This option is helpful when libraries have been installed in some nonstandard location, and you do not wish to make the loader search down those paths when the executable is run at another site. Compare with `-Rpaths`.

See the *Fortran Programming Guide* chapter on *Libraries* for more information.

`-O [n]`

Specify optimization level.

◆ **SPARC: 77/90 x86:77**

n can be 1, 2, 3, 4, or 5. No space is allowed between `-O` and *n*.

If `-O[n]` is not specified, only a very basic level of optimization limited to local common subexpression elimination and dead code analysis is performed. A program's performance may be significantly improved when compiled with an optimization level than without optimization. Use of `-O` (which implies `-O3`) or `-fast` (which implies `-O4`) is recommended for most programs.

Each `-On` level includes the optimizations performed at the levels below it. Generally, the higher the level of optimization a program is compiled with, the better runtime performance obtained. However, higher optimization levels may result in increased compilation time and larger executable files.

Debugging with `-g` does not suppress `-On`, but `-On` limits `-g` in certain ways; this is described on "`-g`" on page 3-34.

The `-O3` and `-O4` options reduce the utility of debugging such that you cannot display variables from `dbx`, but you can still use the `dbx where` command to get a symbolic traceback.

For SPARC: If the optimizer runs out of memory, it attempts to proceed over again at a lower level of optimization, resuming compilation of subsequent routines at the original level.

For details on optimization, see the *Fortran Programming Guide* chapters *Performance Profiling*, and *Performance and Optimization*.

`-O`

This is equivalent to `-O3`.

`-O1`

Provides a minimum of statement-level optimizations.

Use if higher levels result in excessive compilation time, or exceed available swap space.

`-O2`

Enables basic block level optimizations.

This level usually gives the smallest code size. (See also `-xspace`.)

`-O3` is preferred over `-O2` unless `-O3` results in unreasonably long compilation time, exceeds swap space, or generates excessively large executable files.

`-O3`

Adds loop unrolling and global optimizations at the function level.

Usually `-O3` generates larger executable files.

`-O4`

Adds automatic inlining of routines contained in the same file.

Usually `-O4` generates larger executable files due to inlining.

The `-g` option suppresses the `-O4` automatic inlining described above.

`-xcrossfile` increases the scope of inlining with `-O4`.

`-O5`

Attempt aggressive optimizations.

Suitable only for that small fraction of a program that uses the largest fraction of compute time. `-O5`'s optimization algorithms take more compilation time, and may also degrade performance when applied to too large a fraction of the source program.

Optimization at this level is more likely to improve performance if done with profile feedback. See `-xprofile=p`.

`-o nm`

Specify the name of the executable file to be written.

◆ **SPARC: 77/90 x86:77**

There must be a blank between `-o` and `nm`. Without this option, the default is to write the executable file to `a.out`. When used with `-c`, `-o` specifies the target `.o` object file; with `-G` it specifies the target `.so` library file.

`-oldldo`

Select “old” list-directed output style.

◆ **SPARC:77 x86:77**

Omit the blank that starts each record for list-directed output. This is a change from f77 releases 1.4 and earlier. The default behavior is to provide that blank, since the Fortran Standard requires it. Note also the `FORM='PRINT'` option of `OPEN`. You can compile parts of a program with `-oldldo` and other parts without it.

`-onetrip`

Enable one trip `DO` loops.

◆ **SPARC: 77/90 x86:77**

Compile `DO` loops such that they are executed at least once. `DO` loops in standard Fortran are not performed at all if the upper limit is smaller than the lower limit, unlike some legacy implementations of Fortran.

`-p`

Compile for profiling with the `prof` profiler.

◆ **SPARC: 77/90 x86:77**

Prepare object files for profiling, see *prof* (1). If you compile and link in separate steps, and if you compile with the `-p` option, then be sure to link with the `-p` option. `-p` with `prof` is provided mostly for compatibility with older systems. `-pg` profiling

with `gprof` is possibly a better alternative. See the *Fortran Programming Guide* chapter on *Performance Profiling* for details.

`-pad[=p]`

Insert padding for efficient use of cache.

◆ SPARC:77 x86:77

This option inserts padding between arrays or character variables if they are static local and not initialized, or in common blocks. The extra padding positions the data to make better use of cache. In either case, the arrays or character variables can not be equivalenced.

For `-pad[=p]`, if *p* is present, it must be either (or both):

<code>local</code>	Add padding between adjacent <i>local</i> variables
<code>common</code>	Add padding between variables in common blocks

Defaults for `-pad`:

- Without the `-pad[=p]` option, the compiler does no padding.
- With `-pad`, but without the `=p`, the compiler does both local and common padding.

The following are equivalent:

- `f77 -pad any.f`
- `f77 -pad=local,common any.f`
- `f77 -pad=common,local any.f`
- `f77 -pad=local -pad=common any.f`
- `f77 -pad=common -pad=local any.f`

The `-pad[=p]` option applies to items that satisfy the following criteria:

- The items are arrays or character variables
- The items are static local or in common blocks

For a definition of local or static variables, see `-stackvar`, “`-stackvar`” on page 3-56.

Restrictions on `-pad=common`:

- Neither the arrays nor the character strings are equivalenced
- If `-pad=common` is specified for compiling a file that references a common block, it must be specified when compiling all files that reference that common block. The option changes the spacing of variables within the common block. If one program unit is compiled with the option and another is not, references to what should be the same location within the common block might reference different locations.
- If `-pad=common` is specified, the declarations of common block variables in different program units must be the same except for the names of the variables. The amount of padding inserted between variables in a common block depends on the declarations of those variables. If the variables differ in size or rank in different program units, even within the same file, the locations of the variables might not be the same.
- If `-pad=common` is specified, `EQUIVALENCE` declarations involving common block variables are flagged as an error.

`-parallel`

Parallelize loops with: `-autopar`, `-explicitpar`, `-depend`

◆ SPARC: 77/90

Parallelize loops chosen automatically by the compiler *and* explicitly specified by user supplied directives. Optimization level is automatically raised to `-O3` if it is lower.

`-g` cancels `-parallel`. Debugging is facilitated by specifying `-g` without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the `dbx` documentation for details.

To improve performance, also specify the `-stackvar` option when using any of the parallelization options, including `-autopar`.

Avoid `-parallel` if you do your own thread management. See the discussion of `-mt` on “`-mt`” on page 3-41.

Parallelization options like `-parallel` are intended to produce executables programs to be run on multiprocessor systems. On a single-processor system, parallelization generally degrades performance.

If you compile and link in separate steps, if `-parallel` appears on the compile command it must also appear on the `ld` link command.

See the *Fortran Programming Guide* chapter *Parallelization* for further information.

`-pentium`

Compile for Intel Pentium.

◆ **x86:77**

Generate code that exploits features available on Intel Pentium compatible computers. Code compiled with `-pentium` does run on 80386 and 80486 hardware, but it may be slower. Use `-xtarget=pentium` rather than `-pentium`.

`-pg`

Compile for profiling with the `gprof` profiler.

◆ **SPARC: 77/90 x86:77**

Compile self-profiling code in the manner of `-p`, but invoke a runtime recording mechanism that keeps more extensive statistics and produces a `gmon.out` file when the program terminates normally. Generate an execution profile by running `gprof`. See the `gprof(1)` man page and the *Fortran Programming Guide* for details.

Library options must be *after* the `.f` and `.o` files (`-pg` libraries are static).

If you compile and link in separate steps, and you compile with `-pg`, then be sure to link with `-pg`.

`-pic`

Compile position-independent code for shared library.

◆ **SPARC: 77/90 x86:77**

This kind of code is for dynamic shared libraries. Each reference to a global datum is generated as a dereference of a pointer in the global offset table. Each function call is generated in program-counter-relative addressing mode through a procedure linkage table.

- The size of the global offset table is limited to 8Kb on SPARC. The size of the table is unlimited on x86.
- Do not mix `-pic` and `-PIC`.

`-pic` is equivalent to `-xcode=pic13`.

There are two nominal performance costs with `-pic` and `-PIC`:

- A routine compiled with either `-pic` or `-PIC` executes a few extra instructions upon entry to set a register to point at the global offset table used for accessing a shared library's global or static variables.
- Each access to a global or static variable involves an extra indirect memory reference through the global offset table. If the compile is done with `-PIC`, there are two additional instructions per global and static memory reference.

When considering the above costs, remember that the use of `-pic` and `-PIC` can significantly reduce system memory requirements, due to the effect of library code sharing. Every page of code in a shared library compiled `-pic` or `-PIC` can be shared by every process that uses the library. If a page of code in a shared library contains even a single `non-pic` (that is, absolute) memory reference, the page becomes nonsharable, and a copy of the page must be created each time a program using the library is executed.

The easiest way to tell whether or not a `.o` file has been compiled with `-pic` or `-PIC` is with the `nm` command:

```
% nm file.o | grep _GLOBAL_OFFSET_TABLE_
U _GLOBAL_OFFSET_TABLE_
```

A `.o` file containing position-independent code contains an unresolved external reference to `_GLOBAL_OFFSET_TABLE_`, as indicated by the letter `U`.

To determine whether to use `-pic` or `-PIC`, use `nm` to identify the number of distinct global and static variables used or defined in the library. If the size of `_GLOBAL_OFFSET_TABLE_` is under 8,192 bytes, you can use `-pic`. Otherwise, you must use `-PIC`.

When building shared dynamic libraries with `-xarch=v9` or `v9a` on 64-bit Solaris 7, the `-pic` or `-PIC` option, or their `-xcode` equivalents, *must* be specified.

`-PIC`

Compile position-independent code, but with 32-bit addresses.

◆ **SPARC: 77/90 x86:77**

This option is similar to `-pic`, but it allows the global offset table to span the range of 32-bit addresses. Use it in those rare cases where there are too many global data objects for `-pic`. Do not mix `-pic` and `-PIC`.

`-PIC` is equivalent to `-xcode=pic32`.

On `x86`, `-PIC` is identical to `-pic`.

When building shared dynamic libraries with `-xarch=v9` or `v9a` on 64-bit Solaris 7, the `-pic` or `-PIC` option, or their `-xcode` equivalents, *must* be specified.

`-Qoption` *pr ls*

Pass options to compilation phase *pr*.

◆ **SPARC: 77/90 x86:77**

Pass the suboption list *ls* to the compilation phase *pr*. There must be blanks separating `Qoption`, *pr*, and *ls*. The `Q` can be uppercase or lowercase. The list is a comma-delimited list of suboptions, with no blanks within the list. Each suboption must be appropriate for that program phase, and can begin with a minus sign.

This option is provided primarily for debugging the internals of the compiler by support staff. Use the `LD_OPTIONS` environment variable to pass options to the linker. See the chapter on linking and libraries in the *Fortran Programming Guide*.

`-QP`

Synonym for `-p`.

◆ **SPARC: 77/90 x86:77**

`-R` *ls*

Build dynamic library search paths into the executable file.

◆ **SPARC: 77/90 x86:77**

With this option, the linker, `ld(1)`, stores a list of dynamic library search paths into the executable file.

ls is a colon-separated list of directories for library search paths. The blank between `-R` and *ls* is optional.

Multiple instances of this option are concatenated together, with each list separated by a colon.

The list is used at runtime by the runtime linker, `ld.so`. At runtime, dynamic libraries in the listed paths are scanned to satisfy any unresolved references.

Use this option to let users run shippable executables without a special path option to find needed dynamic libraries.

Building an executable file using `-Rpaths` adds directory paths to a default path that is always searched last:

Standard Default Path: `/opt/SUNWspr/lib`

For more information, see the *Libraries* chapter in the *Fortran Programming Guide*, and the Solaris *Linker and Libraries Guide*.

`-r8`

Double default byte size for REAL, INTEGER, DOUBLE and COMPLEX.

◆ SPARC:77 x86:77

`-r8` promotes the default byte size for REAL, INTEGER, DOUBLE, and COMPLEX variables declared *without an explicit byte size* as follows:

TABLE 3-17 Default Data Sizes and `-r8` (Bytes)

Without <code>-r8</code> option		With <code>-r8</code> option	
Data Type	default	SPARC	x86
INTEGER	4	8	8
REAL	4	8	8
DOUBLE	8	16	8

This option applies to variables, parameters, constants, and functions.

Also, LOGICAL is treated as INTEGER, COMPLEX as two REALS, and DOUBLE COMPLEX as two DOUBLES.

`-dbl` and `-r8` can be expressed in terms of the more general `-xtypemap=` option:

■ On SPARC:

`-dbl same as: -xtypemap=real:64,double:128,integer:64 -r8 same as: -xtypemap=real:64,double`

These options promote default DOUBLE PRECISION data to QUAD PRECISION (128 bits). This may be unwanted and may cause performance degradation.

`-xtypemap=real:64,double:64,integer:64` might be more appropriate than `-r8` in these cases.

■ On x86:

-dbl *same as*: -xtypemap=real:64,double:64,integer:64 -r8 *same as*: -xtypemap=real:64,double

- For all of the floating point data types, -dbl works the same as -r8; using both -r8 and -dbl produces the same results as using only -dbl.
- For INTEGER and LOGICAL data types, -dbl differs from -r8:
 - -dbl allocates 8 bytes, and does 8-byte arithmetic
 - -r8 allocates 8 bytes, and does only 4-byte arithmetic (“mixed”)

In general, if you compile *one* subprogram with -r8, then be sure to compile *all* subprograms of that program with -r8. This also important with programs communicating through unformatted I/O files — if one program is compiled with -r8, then the other program must be similarly compiled. Be also aware that this option alters the default data size of function names, including calls to library functions, unless the function name is typed explicitly with a data size.

The impact on runtime performance may be great. With -r8, an expression like `float = 15.0d0*float` is evaluated in quadruple precision due to the declaration of the constant.

If you select both -r8 and -i2, the results are unpredictable.

-reduction

Recognize reduction operations in loops.

◆ SPARC: 77/90

Analyze loops for reduction operations during automatic parallelization. There is potential for roundoff error with the reduction.

A loop that transforms the elements of an array into a single scalar value is called a *reduction operation*. For example, summing the elements of a vector is a typical reduction operation. Although these operations violate the criteria for parallelizability, the compiler can recognize them and parallelize them as special cases when -reduction is specified. See the *Fortran Programming Guide* chapter *Parallelization* for information on reduction operations recognized by the compilers.

This option applies only with the automatic parallelization options -autopar or -parallel. It is ignored otherwise. Explicitly parallelized loops are not analyzed for reduction operations.

Example: Automatically parallelize with *reduction*:

```
demo% f77 -parallel -reduction any.f
```

-S

Compile and only generate assembly code.

◆ **SPARC: 77/90 x86:77**

Compile the named programs and leave the assembly-language output on corresponding files suffixed with `.s`. No `.o` file is created.

-S

Strip the symbol table out of the executable file.

◆ **SPARC: 77/90 x86:77**

This option makes the executable file smaller and more difficult to reverse engineer. However, this option inhibits debugging with `dbx` or other tools, and overrides `-g`.

-sb

Produce table information for the Sun WorkShop source code browser.

◆ **SPARC: 77/90 x86:77**

See *Using Sun WorkShop* for more information.

-sbfast

Produce *only* source code browser tables.

◆ **SPARC:77 x86:77**

Produce *only* table information for the Sun WorkShop source code browser and stop. Do not assemble, link, or make object files.

-silent

Suppress compiler messages.

◆ **SPARC:77 x86:77**

Use this option to suppress non-essential messages from the compiler; error and warning messages are still issued. The default is to show file and entry names as they are reached during the compilation.

`-stackvar`

Force all local variables to be allocated on the memory stack.

◆ **SPARC: 77/90 x86:77**

Allocate all the *local* variables and arrays in a routine onto the memory stack, unless otherwise specified. This option makes them *automatic*, rather than *static*, and provides more freedom to the optimizer for parallelizing a `CALL` in a loop.

Use of `-stackvar` is recommended with any of the parallelization options.

Variables and arrays are local, unless they are:

- Arguments in a `SUBROUTINE` or `FUNCTION` statement (already on stack)
- Global items in a `COMMON` or `SAVE`, or `STATIC` statement
- Items initialized in a type statement or `DATA` statement, such as: `REAL X/8.0/`
or `DATA X/8.0/`

Initializing a local variable in a `DATA` statement after an executable reference to that variable is flagged as an error when `-stackvar` is used:

```
demo% cat stak.f
      real x
      x = 1.
      t = 0.
      print*, t
      data x/3.0/
      print *,x+t
      end

demo% f77 -o stak -stackvar stak.f
stak.f:
MAIN:
"stak.f", line 5: Error: attempt to initialize an automatic
variable: x
```

Putting large arrays onto the stack with `-stackvar` can overflow the stack causing segmentation faults. Increasing the stack size may be required.

There are two stacks:

- The whole program has a *main* stack.

- Each thread of a multi-threaded program has a *thread* stack.

The default stack size is about 8 Megabytes for the main stack and 256 KBytes for each thread stack. The `limit` command (with no parameters) shows the current main stack size. If you get a segmentation fault using `-stackvar`, you might try doubling the main stack size at least once.

Example: Show the current *main* stack size:

```
demo% limit
cputime      unlimited
filesize    unlimited
datasize     523256 kbytes
stacksize    8192 kbytes      <-----
coredumpsize unlimited
descriptors  64
memorysize   unlimited
demo%
```

Example: Set the *main* stack size to 64 Megabytes:

```
demo% limit stacksize 65536
```

Example: Set each *thread* stack size to 8 Megabytes:

```
demo% setenv STACKSIZE 8192
```

For further information of the use of `-stackvar` with parallelization, see the *Parallelization* chapter in the *Fortran Programming Guide*. See `cs(1)` for details on the `limit` command.

`-stop_status=yn`

Permit `STOP` statement to return an integer status value.

◆ SPARC:77 x86:77

`yn` is either `yes` or `no`. The default is `no`.

With `-stop_status=yes`, a `STOP` statement may contain an integer constant. That value will be passed to the environment as the program terminates:

```
STOP 123
```

The value must be in the range 0 to 255. Larger values are truncated and a run-time message issued. Note that

STOP '*stop string*'

is still accepted and returns a status value of 0 to the environment, although a compiler warning message will be issued.

The environment status variable is \$status for the C shell `csh`, and \$? for the Bourne and Korn shells, `sh` and `ksh`.

-temp=*dir*

Define directory for temporary files.

◆ SPARC: 77/90 x86:77

Set directory for temporary files used by the compiler to be *dir*. No space is allowed within this option string. Without this option, the files are placed in the `/tmp` directory.

-time

Time each compilation phase.

◆ SPARC: 77/90 x86:77

The time spent and resources used in each compiler pass is displayed.

-U

Recognize upper and lower case in source files.

◆ SPARC:77 x86:77

Do not treat uppercase letters as equivalent to lowercase. The default is to treat uppercase as lowercase except within character-string constants. With this option, the compiler treats `Delta`, `DELTA`, and `delta` as different symbols.

Portability and mixing Fortran with other languages may require use of `-U`. These are discussed in the *Fortran Programming Guide*. (Note that `f90` does not have this option, always treating upper and lower case as equivalent.)

-u

Report undeclared variables.

◆ SPARC:77 x86:77

Make the default type for all variables be *undeclared* rather than using Fortran implicit typing. This option warns of undeclared variables, and does not override any IMPLICIT statements or explicit *type* statements.

-unroll=*n*

Enable unrolling of DO loops where possible.

◆ SPARC: 77/90 x86:77

n is a positive integer. The choices are:

- *n*=1 inhibits all loop unrolling.
- *n*>1 suggests to the optimizer that it attempt to unroll loops *n* times.

Loop unrolling generally improves performance, but will increase the size of the executable file. For more information on this and other compiler optimizations, see the *Performance and Optimization* chapter in the *Fortran Programming Guide*. See also the discussion of the UNROLL directive on “The UNROLL Directive ” on page 2-9.

-V

Show name and version of each compiler pass.

◆ SPARC: 77/90 x86:77

This option prints the name and version of each pass as the compiler executes:

```
demo% f77 -o scalc -fast -autopar -v scalc.f
f77: WorkShop Compilers 5.0 dev 01 July 1998 FORTRAN 77 5.0
f77pass1: WorkShop Compilers 5.0 dev 01 July 1998 FORTRAN 77 5.0
scalc.f:
  MAIN scalc:
    initial:
    calc1:
    calc2:
    calc3:
iropt: WorkShop Compilers 5.0 dev 01 July 1998
cg: WorkShop Compilers 5.0 dev 01 July 1998
ld: Software Generation Utilities (SGU) SunOS/ELF (LK--1.4 (S/I))
```

This information will be helpful when discussing problems with Sun service engineers.

-V

Verbose mode – show details of each compiler pass.

◆ **SPARC: 77/90 x86:77**

Like **-V**, shows the name of each pass as the compiler executes, and details the options and environment variables used by the driver.

-vax=V

Specify choice of VMS Fortran extensions enabled.

◆ **SPARC:77 x86:77**

v must be a comma-separated list of at least one suboption. Negatives may be constructed by prefixing each suboption keyword by **no%** (as in **no%logical_name**).

The primary options are **-vax=align** and **-vax=misalign**.

-vax=align selects all the suboptions without allowing misaligned data. This is the behavior of the **-x1** option prior to **f77** release 3.0.1.

-vax=misalign selects all the suboptions and allows misaligned data. This is the behavior of the **-x1** option with **f77** releases 3.0.1, 4.0, 4.2, and 5.0.

The table below lists suboptions that can be individually selected.

TABLE 3-18 **-vax=** Suboptions

-vax=	Affect
blank_zero	Treat blank in a numeric field as zero.
bslash	Allow backslash ('\') in character constants.
debug	Allow VMS Fortran 'D' debugging statements.
logical_name	Allow VMS Fortran style logical file names.

TABLE 3-18 `-vax=` Suboptions (continued)

<code>-vax=</code>	Affect
<code>oct_const</code>	Allow double quote character to signify octal constants.
<code>param</code>	Allow non-standard form of PARAMETER statement.
<code>rsize</code>	Allow unformatted record size in words rather than bytes.
<code>struct_align</code>	Align structures as in VMS Fortran.

`%all` and `%none` can also be used to select all or none of these suboptions.

Sub- options accumulate from left to right. For example, to enable all but one feature: `-vax=%all,no%rsize`

See also `-xl` and `-misalign`.

`-vpara`

Show verbose parallelization messages.

◆ **SPARC: 77/90**

As the compiler analyzes loops explicitly marked for parallelization with directives, it issues a warning message about certain data dependencies it detects; but the loop will still be parallelized.

Example: `-vpara` for verbose parallelization warnings:

```
demo% f77 -explicitpar -vpara any.f
any.f:
  MAIN any:
"any.f", line 11: Warning: the loop may have parallelization inhibiting reference
```

`-W`

Suppress warning messages.

◆ **SPARC: 77/90 x86:77**

This option suppresses most warning messages. However, if one option overrides all or part of an option earlier on the command line, you do get a warning.

Example: `-w` still allows some warnings to get through:

```
demo% f77 -w -fast -silent -O4 any.f
f77: Warning: --O4 overwrites previously set optimization level of --O3
demo%
```

For `f90`: Individual levels from 0 to 4 can be specified: `-w0` suppresses the least messages while `-w4` suppresses most warning, `-w` is equivalent to `-w0`.

`-xa`

Synonym for `-a`.

◆ **SPARC: 77/90 x86:77**

`-xarch=a`

Specify target architecture instruction set.

◆ **SPARC: 77/90 x86:77**

Target architectures specified by keyword `a` are:

TABLE 3-19 `-xarch` Architecture Keywords

<i>On SPARC:</i>	generic, v7, v8a, v8, v8plus, v8plusa, v9, v9a
<i>On x86:</i>	generic, 386, pentium_pro

Although this option can be used alone, it is part of the expansion of the `-xtarget` option; it is provided to allow overriding the `-xarch` value implied by a specific `-xtarget` option.

This option limits the instructions generated to those of the specified architecture, and *allows* the specified set of instructions. It does not guarantee that target-specific instructions are used.

If this option is used with optimization, the appropriate choice can provide good performance of the executable on the specified architecture. An inappropriate choice can result in serious degradation of performance.

For SPARC Platforms:

SPARC architectures `v7`, `v8`, and `v8a` are all binary compatible. `v8plus` and `v8plusa` are binary compatible with each other and forward, but not backward compatible with earlier architectures.

`v9` and `v9a` are only available on 64-bit Solaris 7 environments, and are binary compatible with each other but not compatible with earlier architectures.

For any particular choice, the generated executable may run much more slowly on earlier architectures.

generic: Get good performance on most systems.

This is the default. This option uses the best instruction set for good performance on most processors without major performance degradation on any of them. With each new release, the definition of “best” instruction set may be adjusted, if appropriate.

v7: Limit the instruction set to V7 architecture.

This option uses the best instruction set for good performance on the V7 architecture, but without the quad-precision floating-point instructions.

This is equivalent to using the best instruction set for good performance on the V8 architecture, but *without*:

- The quad-precision floating-point instructions
- The integer `mul` and `div` instructions
- The `fsmuld` instruction

Examples: SPARCstation 1, SPARCstation 2

v8a: Limit the instruction set to the V8a version of the V8 architecture.

By definition, V8a means the V8 architecture, but without:

- The quad-precision floating-point instructions
- The `fsmuld` instruction

This option uses the best instruction set for good performance on the V8a architecture.

Example: Any machine based on the MicroSPARC I chip architecture

v8: Limit the instruction set to V8 architecture.

This option uses the best instruction set for good performance on the V8 architecture, but without quad-precision floating-point instructions.

Example: SPARCstation 10

v8plus: Limit instructions to the V8plus version of the V9 architecture.

By definition, V8plus means the V9 architecture, except:

- Without the quad-precision floating-point instructions
- Limited to the 32-bit subset defined by the V8plus specification
- Without the VIS instructions

This option uses the best instruction set for good performance on the V8plus chip architecture. In V8plus, a system with the 64-bit registers of V9 runs in 32-bit addressing mode, but the upper 32 bits of the `ix` and `lx` registers must not affect program results.

Example: Any machine based on the UltraSPARC chip architecture

Use of `-xarch=v8plus` causes the `.o` file to be marked as a V8+ binary. Such binaries will not run on a V7 or V8 machine.

v8plusa: Limit the instruction set to the V8plusa architecture variation.

By definition, V8plusa, means the V8plus architecture, plus:

- The UltraSPARC-specific instructions
- The VIS instructions

This option uses the best instruction set for good performance on the UltraSPARC™ architecture, but limited to the 32-bit subset defined by the V8plus specification.

Example: Any machine based on the UltraSPARC chip architecture

Use of `-xarch=v8plusa` also causes the `.o` file to be marked as a Sun-specific V8plus binary. Such binaries will not run on a V7 or V8 machine.

v9: Limit instruction set to the SPARC-V9 architecture.

The resulting `.o` object files are in 64-bit ELF format and can only be linked with other object files in the same format. The resulting executable can only be run on a 64-bit SPARC processor running 64-bit Solaris 7 with the 64-bit kernel. Compiling with this option uses the best instruction set for good performance on the V9 SPARC architecture, but without the use of quad-precision floating-point instructions. `-xarch=v9` is only available when compiling in the 64-bit Solaris 7 environment.

v9a: Limits instruction set to the SPARC-V9 architecture with VIS and UltraSPARC.

This option adds the Visual Instruction Set (VIS) and extensions specific to UltraSPARC processors.

The resulting `.o` object files are in 64-bit ELF format and can only be linked with other object files in the same format. The resulting executable can only be run on a 64-bit SPARC processor running 64-bit Solaris 7 with the 64-bit kernel. Compiling with this option uses the best instruction set for good performance on the V9 SPARC

architecture, but without the use of quad-precision floating-point instructions.
-xarch=v9a is only available when compiling in the 64-bit Solaris 7 environment.

For x86 Platforms:

generic and 386 are equivalent in this release.

pentium_pro directs the compiler to issue instructions for the x86 PentiumPro chip.

-xautopar

Synonym for -autopar.

◆ **SPARC: 77/90**

-xcache=c

Define cache properties for the optimizer.

◆ **SPARC: 77/90**

c must be one of the following:

- generic
- *s1/l1/a1*
- *s1/l1/a1:s2/l2/a2*
- *s1/l1/a1:s2/l2/a2:s3/l3/a3*

The *si/li/ai* are defined as follows:

si The size of the data cache at level *i*, in kilobytes

li The line size of the data cache at level *i*, in bytes

ai The associativity of the data cache at level *i*

This option specifies the cache properties that the optimizer can use. It does not guarantee that any particular cache property is used.

Although this option can be used alone, it is part of the expansion of the -xtarget option; it is provided to allow overriding an -xcache value implied by a specific -xtarget option.

TABLE 3-20 `-xcache` Values

Value	Meaning
<code>generic</code>	Define the cache properties for good performance on most SPARC processors without any major performance degradation. This is the default.
<code>s1/l1/a1</code>	Define level 1 cache properties.
<code>s1/l1/a1:s2/l2/a2</code>	Define levels 1 and 2 cache properties.
<code>s1/l1/a1:s2/l2/a2:s3/l3/a3</code>	Define levels 1, 2, and 3 cache properties

Example: `-xcache=16/32/4:1024/32/1` specifies the following:

A Level 1 cache has: 16K bytes, 32 byte line size, 4-way associativity.

A Level 2 cache has: 1024K bytes, 32 byte line size, direct mapping associativity.

`-xcg89`

Synonym for `-cg89`.

◆ SPARC: 77/90

`-xcg92`

Synonym for `-cg92`.

◆ SPARC: 77/90

`-xchip=c`

Specify target processor for the optimizer.

◆ SPARC: 77/90 x86:77

This option specifies timing properties by specifying the target processor.

Although this option can be used alone, it is part of the expansion of the `-xtarget` option; it is provided to allow overriding a `-xchip` value implied by the a specific `-xtarget` option.

Some effects of `-xchip=c` are:

- Instruction scheduling
- The way branches are compiled
- The instructions to use in cases where semantically equivalent alternatives are available

The following table lists the valid `-xchip` values:

TABLE 3-21 Valid `-xchip` Values

	Value	Optimize for:
SPARC:	generic	good performance on most SPARC processors.
	old	pre-SuperSPARC™ processors.
	super	the SuperSPARC chip.
	super2	the SuperSPARC II chip.
	micro	the MicroSPARC™ chip.
	micro2	the MicroSPARC II chip.
	hyper	the HyperSPARC™ chip.
	hyper2	the HyperSPARC II chip.
	powerup	the Weitek™ PowerUp™ chip.
	ultra	the UltraSPARC™ chip.
	ultra2	the UltraSPARC II™ chip.
	ultra2i	the UltraSPARC III™ chip.
x86:	generic	good performance on most x86 processors
	386	Intel 386
	486	Intel 486
	pentium	Intel Pentium
	pentium_pro	Intel Pentium Pro

`-xcode=code`

Specify code address space on SPARC platforms.

◆ SPARC: 77/90

The values for *code* are:

<code>abs32</code>	Generate 32-bit absolute addresses. Code+data+bss size is limited to 2**32 bytes. This is the default on 32-bit platforms: <code>-xarch=generic, v7, v8, v8a, v8plus, v8plusa</code>
<code>abs44</code>	Generate 44-bit absolute addresses. Code+data+bss size is limited to 2**44 bytes. Available only on 64-bit platforms: <code>-xarch=v9, v9a</code>
<code>abs64</code>	Generate 64-bit absolute addresses. Available only on 64-bit platforms: <code>-xarch=v9, v9a</code>
<code>pic13</code>	Generate position-independent code (small model). Equivalent to <code>-pic</code> . Permits references to at most 2**11 unique external symbols on 32-bit platforms, 2**10 on 64-bit platforms.
<code>pic32</code>	Generate position-independent code (large model). Equivalent to <code>-PIC</code> . Permits references to at most 2**30 unique external symbols on 32-bit platforms, 2**29 on 64-bit platforms.

The defaults (not specifying `-xcode=code` explicitly) are:

`-xcode=abs32` on SPARC V8 and V7 platforms. `-xcode=abs64` on SPARC and UltraSPARC V9 (`-xarch=v9` or `v9a`)

When building shared dynamic libraries with `-xarch=v9` or `v9a` and the 64-bit Solaris 7 environment, `-xcode=pic13` or `-xcode=pic32` (or `-pic` or `-PIC`) *must* be specified.

`-xcommonchk [= { no | yes }]`

Enable runtime checking of common block inconsistencies.

◆ SPARC: 77/90

This option provides a debug check for common block inconsistencies in programs using `TASK COMMON` and parallelization. (See the discussion of the `TASK COMMON` directive in the *Parallelization* chapter in the *Fortran Programming Guide*.)

The default is `-xcommonchk=no`; runtime checking for common block inconsistencies is disabled because it will degrade performance. Use it only during program development and debugging, and not for production-quality programs.

Compiling with `-xcommonchk=yes` enables runtime checking. If a common block declared in one source program unit as a regular common block appears somewhere else on a `TASK COMMON` directive, the program will stop with an error message indicating the first such inconsistency.

`-xcrossfile [=n]`

Enable optimization and inlining across source files.

◆ SPARC:77

If specified, *n* may be 0, or 1.

Normally, the scope of the compiler's analysis is limited to each separate file on the command line. For example, `-O4`'s automatic inlining is limited to subprograms defined and referenced within the same source file.

With `-xcrossfile`, the compiler analyzes all the files named on the command line as if they had been concatenated into a single source file.

`-xcrossfile` is only effective when used with `-O4` or `-O5`.

Cross-file inlining creates a possible source file interdependence that would not normally be there. If any file in a set of files compiled together with `-xcrossfile` is changed, then all files must be recompiled to insure that the new code is properly inlined. See the discussion of inlining on "`-inline=f1[,...fn]`" on page 3-37.

The default, without `-xcrossfile` on the command line, is `-xcrossfile=0`, and no cross-file optimizations are performed. To enable the first level of cross-file optimizations, specify `-xcrossfile` (equivalent to `-xcrossfile=1`).

`-xdepend`

Synonym for `-depend`.

◆ SPARC: 77/90

`-xexplicitpar`

Synonym for `-explicitpar`.

◆ **SPARC: 77/90**

`-xF`

Allow function-level reordering by the Sun WorkShop Analyzer.

◆ **SPARC:77 x86:77**

Allow the reordering of functions (subprograms) in the core image using the compiler, the Analyzer and the linker. If you compile with the `-xF` option, then run the Analyzer, you can generate a map file that optimizes the ordering of the functions in memory depending on how they are used together. A subsequent link to build the executable file can be directed to use that map by using the linker `-Mmapfile` option. It places each function from the executable file into a separate section.

Reordering the subprograms in memory is useful only when the application text page fault time is consuming a large percentage of the application time. Otherwise, reordering may not improve the overall performance of the application. The Analyzer is part of the Sun WorkShop. See *Using Sun WorkShop* and *Analyzing Program Performance with Sun WorkShop* for further information on the Analyzer.

`-xhelp=h`

Show summary help information on options or README file.

◆ **SPARC: 77/90 x86:77**

The *h* is either `readme` or `flags`.

`readme`: Show the online README file for this release of the compiler. `flags`: Show the compiler flags (options).

`-xhelp=flags` is a synonym for `-help`.

`-xildoff`

Turn off the Incremental Linker.

◆ **SPARC: 77/90**

This forces the use of the standard linker, ld.

This option is the default if you do *not* use the `-g` option. It is also the default if you use `-G` or name any source file on the command line.

Override this default by using the `-xildon` option.

`-xildon`

Turn on the Incremental Linker.

◆ SPARC: 77/90

Turn on the Incremental Linker and force the use of `ild` in incremental mode.

This option is the default if you use `-g` and do *not* use `-G`, and do *not* name any source file on the command line.

Override this default by using the `-xildoff` option.

See the section on `ild` in the *Debugging a Program With Sun WorkShop* guide.

`-xinline=f1[,...,fn]`

Synonym for `-inline=f1[,...,fn]`.

◆ SPARC: 77/90 x86:77

`-xl[d]`

Enable more VMS Fortran extensions.

◆ SPARC:77 x86:77

`-xl`: Enable the compiler to accept more VMS Fortran extensions. This is a macro that is translated to `-vax=misalign`, and provides the language features that are listed later in this description. See the description of `-vax=`, “`-vax=v`” on page 3-60.

Although most VMS features are accepted automatically by `f77` without any special options, you must use the `-xl` option for a few VMS extensions.

In general, you need the `-xl` option if a source statement can be interpreted as either a VMS feature or an `f77` or `f90` feature, and you want the VMS feature. In this case, the `-xl` option forces the compiler to interpret it the VMS way.

This option enables the following VMS language features:

- Unformatted record size in words rather than bytes (-x1)
- VMS style logical file names (-x1)
- Quote (") character introducing octal constants (-x1)
- Backslash (\) as ordinary character within character constants (-x1)
- Nonstandard form of the PARAMETER statement (-x1)
- Alignment of structures as in VMS. (-x1)
- Debugging lines as comment lines or Fortran statements (-x1d)

Use -x1 to get VMS alignment if your program has some detailed knowledge of how VMS structures are implemented.

Use -x1d to cause compilation of debugging comments (D or d in column one). Without the -x1d option, they remain comments only. (There is no space between -x1 and d.)

Programs that share structures with C routines should not use -x1.

See the *Fortran Library Reference* for information on the VMS libraries. See also the chapter on VMS language extensions in the *FORTTRAN 77 Language Reference* that the f77 compiler automatically recognizes.

-xlibmil

Synonym for -libmil.

- ◆ SPARC: 77/90 x86:77

-xlibmopt

Use library of optimized math routines.

- ◆ SPARC: 77/90

Use selected math routines optimized for speed. This option usually generates faster code. It may produce slightly different results; if so, they usually differ in the last bit. The order on the command line for this library option is not significant.

-xlic_lib=*libs*

Link with the specified Sun licensed libraries.

- ◆ SPARC: 77/90 x86:77

Specifies a comma-separated list of libraries to link with. For example:

```
f77 -o pgx --fast pgx.f --xlic_lib=sunperf
```

As with `-l`, this option should appear on the command line after all source and object file names.

`-xlicinfo`

Show license server information.

◆ SPARC: 77/90 x86:77

Use this option to return license information about the licensing system—in particular, the name of the license server and the user ID for each of the users who have licenses checked out.

Generally, with this option, no compilation takes place, and a license is not checked out. This option is normally used alone with no other options. However, if a conflicting option is used, then the last one on the command line prevails, and there is a warning.

`-Xlist[X]`

Produce listings and do global program checking (GPC).

◆ SPARC: 77/90 x86:77

Use this option to find potential programming bugs. It invokes an extra compiler pass to check for consistency in subprogram call arguments, common blocks, and parameters, across the global program. The option also generates a line-numbered listing of the source code, including a cross reference table. The error messages issued by the `-Xlist` options are advisory warnings and do not prevent the program from being compiled and linked.

Note - Be sure to correct all syntax errors in the source code before compiling with a `-Xlist` global program checking output. GPC can produce unpredictable reports when run on a source code with syntax errors.

Example: Check across routines for consistency:

```
demo% f77 -Xlist fil.f
```

The above example writes the following to the output file `fil.lst`:

- A line-numbered source listing (default)
- Error messages (embedded in the listing) for inconsistencies across routines
- A cross reference table of the identifiers (default)

By default, the listings are written to the file `name.lst`, where `name` is taken from the first listed source file on the command line.

A number of sub-options provide further flexibility in the selection of actions. These are specified by suffixes to the main `-xlist` option, as shown in the following table

TABLE 3-22 `-xlist` Suboptions (Not all options are available with `f90`)

Option	Feature
<code>-Xlist</code>	Show errors, listing, and cross reference table
<code>-Xlistc</code>	Show call graphs and errors (<i>f77 only</i>)
<code>-XlistE</code>	Show errors
<code>-Xlisterr[<i>nnn</i>]</code>	Suppress error <i>nnn</i> messages
<code>-Xlistf</code>	Show errors, listing, and cross references, but no object files (<i>f77 only</i>)
<code>-Xlistfndir</code>	Put <code>.fn</code> files in directory <i>dir</i> , which must already exist (<i>f77 only</i>)
<code>-Xlisth</code>	Terminate compilation if errors detected (<i>f77 only</i>)
<code>-XlistI</code>	Analyze <code>#include</code> and <code>INCLUDE</code> files as well as source files
<code>-XlistL</code>	Show listing and errors only
<code>-Xlistln</code>	Set page length to <i>n</i> lines
<code>-Xlisto <i>name</i></code>	Rename report file to <i>name.lst</i>
<code>-Xlists</code>	Suppress unreferenced names from the cross-reference table (<i>f77 only</i>)
<code>-Xlistvn</code>	Set checking level to <i>n</i> (1,2,3, or 4) – default is 2 (<i>f77 only</i>)
<code>-Xlistw[<i>nnn</i>]</code>	Set width of output line to <i>nnn</i> columns – default is 79

TABLE 3-22 `-Xlist` Suboptions (Not all options are available with f90) (continued)

Option	Feature
<code>-Xlistwar[nnn]</code>	Suppress warning <i>nnn</i> messages
<code>-XlistX</code>	Show cross-reference table and errors

Options `-Xlistc`, `-Xlistf`, `-Xlistflndir`, `-Xlisth`, `-Xlists`, and `-Xlistvn` are not available with f90.

See the *Fortran Programming Guide* chapter *Program Analysis and Debugging* for details.

`-xloopinfo`

Synonym for `-loopinfo`.

◆ SPARC: 77/90

`-xmaxopt [=n]`

Enable optimization pragma and set maximum optimization level.

◆ SPARC:77 x86:77

n has the value 1 through 5 and corresponds to the optimization levels of `-O1` through `-O5`. If not specified, the compiler uses 5.

This option enables the `C$PRAGMA SUN OPT=n` directive when it appears in the source input. Without this option, the compiler treats these lines as comments.

If such a pragma directive appears with an optimization level greater than the maximum level on the `-xmaxopt` flag, the compiler uses `-xmaxopt` level.

`-xnolib`

Synonym for `-nolib`.

◆ SPARC: 77/90 x86:77

`-xnoibmil`

Synonym for `-noibmil`.

◆ **SPARC: 77/90 x86:77**

`-xnoibmopt`

Do not use fast math library.

◆ **SPARC: 77/90 x86:77**

Use with `-fast` to override linking the optimized math library:

```
f77 -fast -xnoibmopt
```

`-xO[n]`

Synonym for `-O[n]`.

◆ **SPARC: 77/90 x86:77**

`-xpad`

Synonym for `-pad`.

◆ **SPARC:77**

`-xparallel`

Synonym for `-parallel`.

◆ **SPARC: 77/90**

`-xpg`

Synonym for `-pg`.

◆ **SPARC: 77/90 x86:77**

`-xpp={ fpp | cpp }`

Select source file preprocessor.

◆ **SPARC: 77/90 x86:77**

The default is `-xpp=fpp`.

The compilers use `fpp(1)` to preprocess `.F` or `.F90` source files. This preprocessor is appropriate for Fortran. Previous versions used the standard C preprocessor `cpp`. To select `cpp`, specify `-xpp=cpp`.

`-xprefetch[={ yes | no }]`

Use prefetch instructions on Ultrasparc II platforms.

◆ **SPARC: 77/90**

Specifying `-xprefetch=yes` enables the compiler to insert prefetch instructions whenever appropriate. This may result in a performance improvement on UltraSPARC II processors.

If not specified, the compiler assumes `-xprefetch=no`. `-xprefetch` specified by itself defaults to `-xprefetch=yes`.

`-xprofile=p`

Collect or optimize with runtime profiling data.

◆ **SPARC: 77/90 x86:77**

p must be one of `collect[:nm]`, `use[:nm]`, or `tcov`. Optimization level must be `-O2` or greater.

`collect[:nm]`

Collect and save execution frequency data for later use by the optimizer with `-xprofile=use`. The compiler generates code to measure statement execution frequency.

The *nm* is the name of the program that is being analyzed. This name is optional. If *nm* is not specified, `a.out` is assumed to be the name of the executable.

At runtime a program compiled with `-xprofile=collect:nm` will create the subdirectory `nm.profile` to hold the runtime feedback information. Data is written

to the file `feedback` in this subdirectory. If you run the program several times, the execution frequency data accumulates in the `feedback` file; that is, output from prior runs is not lost.

`use[:nm]`

Use execution frequency data to optimize strategically.

As with `collect:nm`, the `nm` is optional and may be used to specify the name of the program.

The program is optimized by using the execution frequency data previously generated and saved in the `feedback` files written by a previous execution of the program compiled with `-xprofile=collect`.

The source files and other compiler options must be exactly the same as used for the compilation that created the compiled program that generated the `feedback` file. If compiled with `-xprofile=collect:nm`, the same program name `nm` must appear in the optimizing compilation: `-xprofile=use:nm`.

`tcov`

Basic block coverage analysis using “new” style `tcov`.

Code instrumentation is similar to that of `-a`, but `.d` files are no longer generated for each source file. Instead, a single file is generated, whose name is based on the name of the final executable. For example, if `stuff` is the executable file, then `stuff.profile/tcovd` is the data file.

When running `tcov`, you must pass it the `-x` option to make it use the new style of data. If not, `tcov` uses the old `.d` files, if any, by default for data, and produces unexpected output.

Unlike `-a`, the `TCOVDIR` environment variable has no effect at compile-time. However, its value is used at program runtime to identify where to create the profile subdirectory.

See the `tcov(1)` man page, the *Performance Profiling* chapter of the *Fortran Programming Guide*, and the *Analyzing Program Performance with Sun WorkShop* manual for more details.

`-xreduction`

Synonym for `-reduction`.

◆ **SPARC: 77/90**

`-xregs=r`

Specify register usage.

◆ **SPARC: 77/90**

r is a comma-separated list that consists of one or more of the following:

[no%]appl, [no%]float.

Where the % is shown, it is a required character.

Example: `-xregs=appl,no%float`

appl: Allow using the application registers.

On SPARC systems, certain registers are described as *application* registers. Using these registers can increase performance because fewer load and store instructions are needed. However, such use can conflict with some old library programs written in assembly code.

The set of application registers depends on the SPARC platform:

`-xarch=v8` or `v8a` — registers *g2*, *g3*, and *g4*

`-xarch=v8` or `v8a` — registers *g2*, *g3*, and *g4*

`-xarch=v8plus` or `v8plusa` — registers *g2*, *g3*, *g4*, and *g5*

`-xarch=v9` or `v9a` — registers *g2* and *g3*

no%appl: Do not use the appl registers.

float: Allow using the floating-point registers as specified in the SPARC ABI.

You can use these registers even if the program contains no floating-point code.

no%float: Do not use the floating-point registers.

With this option, a source program cannot contain any floating-point code.

The default is: `-xregs=appl,float`.

—XS

Allow debugging by `dbx` without `object (.o)` files.

◆ **SPARC: 77/90 x86:77**

With `-xs`, if you move executables to another directory, then you can use `dbx` and ignore the `object (.o)` files. Use this option when you cannot keep the `.o` files.

- The compiler passes `-s` to the assembler and then the linker places all symbol tables for `dbx` in the executable file.
- This way of handling symbol tables is the older way. It is sometimes called *no auto-read*.

- The linker links more slowly, and `dbx` initializes more slowly.

Without `-xs`, if you move the executables, you must move both the source files and the object (`.o`) files, or set the path with either the `dbx pathmap` or `use` command.

- This way of handling symbol tables is the newer and default way of loading symbol tables. It is sometimes called *auto-read*.
- The symbol tables are distributed in the `.o` files so that `dbx` loads the symbol table information only if and when it is needed. Hence, the linker links faster, and `dbx` initializes faster.

`-xsafe=mem`

Assume no memory-based traps.

- ◆ **SPARC: 77/90**

Using this option allows the compiler to assume no memory-based traps occur. It grants permission to use the speculative load instruction on V9 machines. It is only effective if `-O5` and `-xarch=v8plus` are also specified.

`-xsb`

Synonym for `-sb`.

- ◆ **SPARC: 77/90 x86:77**

`-xsbfast`

Synonym for `-sbfast`.

- ◆ **SPARC:77 x86:77**

`-xspace`

Do not allow optimizations to increase code size.

- ◆ **SPARC: 77/90**

Do no optimizations that increase the code size.

Example: Do not unroll or parallelize loops if it increases code size.

`-xtarget=t`

Specify target platform for optimization.

◆ SPARC: 77/90 x86:77

Specify the target platform for the instruction set and optimization.

t must be one of: `native`, `generic`, *platform-name*.

The `-xtarget` option permits a quick and easy specification of the `-xarch`, `-xchip`, and `-xcache` combinations that occur on real platforms. The only meaning of `-xtarget` is in its expansion.

The performance of some programs may benefit by providing the compiler with an accurate description of the target computer hardware. When program performance is critical, the proper specification of the target hardware could be very important. This is especially true when running on the newer SPARC processors. However, for most programs and older SPARC processors, the performance gain is negligible and a `generic` specification is sufficient.

`native`: Optimize performance for the host platform.

The compiler generates code optimized for the host platform. It determines the available architecture, chip, and cache properties of the machine on which the compiler is running.

`generic`: Get the best performance for generic architecture, chip, and cache.

The compiler expands `-xtarget=generic` to:

```
-xarch=generic -xchip=generic -xcache=generic
```

This is the default value.

platform-name: Get the best performance for the specified platform.

See Appendix D, Appendix D, for a complete list of current SPARC platform names accepted by the compilers. For example, `-xtarget=ultra2i`

For x86: `-xtarget=` accepts:

- `generic` or `native`
- `386` (equivalent to `-386` option) or `486` (equivalent to `-486` option)
- `pentium` (equivalent to `-pentium` option) or `pentium_pro`

`-xtime`

Synonym for `-time`.

◆ SPARC: 77/90 x86:77

`-xtypemap=spec`
Specify default data mappings.

◆ **SPARC:77 x86:77**

This option provides a flexible way to specify the byte sizes for default data types. Use of this option is preferred over `-dbl` and `-r8`.

The syntax of the specification string *spec* is:

`real: size, double: size, integer: size.`

No parts of the specification string can be left out. The accepted data sizes are 64, 128, for real and double, and mixed for integer. For example: “`-xtypemap=real:64, double:64, integer:64`”

This option applies to all variables declared with default specifications (without explicit byte sizes), as in `REAL XYZ`.

The allowable combinations on each platform are:

TABLE 3-23 Allowed `-xtypemap=` mappings

SPARC	x86
<code>real:64</code>	<code>real:64</code>
<code>double:64</code>	<code>double:64</code>
<code>double:128</code>	-
<code>integer:64</code>	<code>integer:64</code>
<code>integer:mixed</code>	<code>integer:mixed</code>

The mapping `integer:mixed` indicates 8 byte integers but only 4 byte arithmetic.

The `-dbl` and `-r8` options have their `-xtypemap` equivalents

■ **SPARC:**

`-dbl same as: -xtypemap=real:64, double:128, integer:64` `-r8 same as: -xtypemap=real:64, double:64, integer:64`

■ **x86:**

`-dbl same as: -xtypemap=real:64, double:64, integer:64` `-r8 same as: -xtypemap=real:64, double:64, integer:64`

There are two additional possibilities on SPARC:

`-xtypemap=real:64, double:64, integer:mixed` `-xtypemap=real:64, double:64, integer:64`

which map both default REAL and DOUBLE to 8 bytes, and may be preferable over the use of `-dbl` or `-r8` because they do not promote DOUBLE PRECISION to QUAD PRECISION.

Note that INTEGER and LOGICAL are treated the same, and COMPLEX is mapped as two REALS. Also, DOUBLE COMPLEX will be treated the way DOUBLE is mapped.

`-xunroll=n`

Synonym for `-unroll=n`.

◆ SPARC: 77/90 x86:77

`-xvector[={yes|no}]`

Enable automatic calls to the SPARC vector library functions.

◆ SPARC: 77/90

With `-xvector=yes`, the compiler is permitted to transform certain math library calls within DO loops into single calls to the equivalent vectorized library routine whenever possible. This could result in a performance improvement for loops with large loop counts.

The compiler defaults to `-xvector=no`. Specifying `-xvector` by itself defaults to `-xvector=yes`.

This option also triggers `-depend`. (Follow `-xvector` with `-nodepend` on the command line to cancel the dependency analysis.)

The compiler will automatically notify the linker to include the `libmvec` and `libc` libraries in the load step if `-xvector` appears. However, to compile and link in separate steps requires specifying `-xvector` on the link step as well to correctly select these necessary libraries.

`-xvpara`

Synonym for `-vpara`.

◆ SPARC:77

`-Zlp`

Compile for loop performance profiling by `looptool`.

◆ SPARC: 77/90

Prepare object files for the loop profiler, `looptool`. The `looptool(1)` utility can then be run to generate loop statistics about the program.

If you compile and link in separate steps, and you compile with `-zlp`, then be sure to link with `-zlp`.

If you compile *one* subprogram with `-zlp`, you need not compile *all* the subprograms of that program with `-zlp`. However, you receive the loop information only for the files compiled with `-zlp`, and no indication that the program includes other files.

Refer to *Analyzing Program Performance With Sun WorkShop* for more information.

`-ztext`

Generate only pure libraries with no relocations.

◆ SPARC: 77/90

Do not make the library if relocations remain.

The general purpose of `-ztext` is verify that a generated library is pure text; instructions are all position-independent code. Therefore, it is generally used with both `-G` and `-pic`.

With `-ztext`, if `ld` finds an incomplete relocation in the *text* segment, then it does not build the library. If it finds one in the *data* segment, then it generally builds the library anyway; the data segment is writable.

Without `-ztext`, `ld` builds the library, relocations or not.

A typical use is to make a library from both source files and object files, where you do not know if the object files were made with `-pic`.

Example: Make library from both source and object files:

```
demo% f77 -G -pic -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

An alternate use is to ask if the code is position-independent already: compile without `-pic`, but ask if it is pure text.

Example: Ask if it is pure text already—even without `-pic`:

```
demo% f77 -G -ztext -o MyLib -hMyLib a.f b.f x.o y.o
```

If you compile with `-ztext` and `ld` does not build the library, then you can recompile without `-ztext`, and `ld` will build the library. The failure to build with `-ztext` means that one or more components of the library cannot be shared; however, maybe some of the other components can be shared. This raises questions of performance that are best left to you, the programmer.

Runtime Error Messages

This appendix describes the error messages generated by the Fortran I/O library, signal handler, and operating system.

Operating System Error Messages

Operating system error messages include system call failures, C library errors, and shell diagnostics. The system call error messages are found in *intro(2)*. System calls made through the Fortran library do not produce error messages directly. The following system routine in the Fortran library calls C library routines which produce an error message:

```
CALL SYSTEM("rm /")  
END
```

The following message is displayed:

```
rm: / directory
```

Signal Handler Error Messages

Before beginning execution of a program, the Fortran library sets up a signal handler (*sigdie*) for signals that can cause termination of the program. *sigdie* prints a

message that describes the signal, flushes any pending output, and generates a core image and a traceback.

Presently, the only arithmetic exception that produces an error message is the INTEGER*2 division with a denominator of zero. All other arithmetic exceptions are ignored.

A signal handler error example follows, where the subroutine SUB tries to access parameters that are not passed to it:

```
CALL SUB()  
END  
SUBROUTINE SUB(I,J,K)  
I=J+K  
RETURN  
END
```

The following error message results:

```
*** Segmentation violation  
Illegal instruction (core dumped)
```

I/O Error Messages (f77)

The error messages in this section are generated by the Fortran 77 I/O library. The error numbers are returned in the IOSTAT variable if the ERR return is taken.

For example, the following program tries to do an unformatted write to a file opened for formatted output:

```
WRITE( 6 ) 1  
END
```

and produces error messages like the following:

```
sue: [1003] unformatted io not allowed  
logical unit 6, named "stdout"  
lately: writing sequential unformatted external IO
```

The following error messages are generated. These same messages are also documented at the end of the man page *perror(3F)*.

If the error number is less than 1000, then it is a *system* error. See *intro* (2).

TABLE A-1 f77 Runtime I/O Messages

Error	Message
1000	error in format Read the error message output for the location of the error in the format. It can be caused by more than 10 levels of nested parentheses or an extremely long format statement.
1001	illegal unit number It is illegal to close logical unit 0. Negative unit numbers are not allowed. The upper limit is $2^{31} - 1$.
1002	formatted io not allowed The logical unit was opened for unformatted I/O.
1003	unformatted io not allowed The logical unit was opened for formatted I/O.
1004	direct io not allowed The logical unit was opened for sequential access, or the logical record length was specified as 0.
1005	sequential io not allowed The logical unit was opened for direct access I/O.
1006	can't backspace file You cannot do a seek on the file associated with the logical unit; therefore, you cannot backspace. The file may be a tty device or a pipe.
1007	off beginning of record You tried to do a left tab to a position before the beginning of an internal input record.
1008	can't stat file The system cannot return status information about the file. Perhaps the directory is unreadable.

TABLE A-1 f77 Runtime I/O Messages (continued)

Error	Message
1009	no * after repeat count Repeat counts in list-directed I/O must be followed by an * with no blank spaces.
1010	off end of record A formatted write tried to go beyond the logical end-of-record. An unformatted read or write also causes this
1011	<Not used>
1012	incomprehensible list input List input has to be as specified in the declaration.
1013	out of free space The library dynamically creates buffers for internal use. You ran out of memory for them; that is, your program is too big.
1014	unit not connected The logical unit was not open.
1015	read unexpected character Certain format conversions cannot tolerate nonnumeric data.
1016	illegal logical input field logical data must be T or F.
1017	'new' file exists You tried to open an existing file with status="new".
1018	can't find 'old' file You tried to open a nonexistent file with status="old".
1019	unknown system error This error should not happen, but..

TABLE A-1 f77 Runtime I/O Messages (continued)

Error	Message
1020	requires seek ability Attempted a seek on a file that does not allow it. I/O operation requiring a seek are direct access, sequential unformatted I/O, and tabbing left.
1021	illegal argument Certain arguments to open and related functions are checked for legitimacy. Often only nondefault forms are checked
1022	negative repeat count The repeat count for list-directed input must be a positive integer.
1023	illegal operation for unit Attempted an I/O operation that is not possible for the device associated with the logical unit. You get this error if you try to read past end-of-tape, or end-of-file.
1024	<Not used>
1025	incompatible specifiers in open Attempted to open a file with the "new" option and the access="append" option, or some other invalid combination.
1026	illegal input for namelist A namelist read encountered an invalid data item.
1027	error in FILEOPT parameter The FILEOPT string in an OPEN statement has bad syntax.
1028	WRITE to readonly file Attempt to write on a unit that was opened for reading only.
1029	READ from writeonly file Attempt to read from a unit that was opened for writing only.

TABLE A-1 f77 Runtime I/O Messages (continued)

Error	Message
1030	overflow converting numeric input Integer input data is too large for the corresponding input variable
1032	exponent overflow on numeric input The floating-point input data is too large to be represented by the corresponding input variable.

I/O Error Messages (f90)

This is a partial list of runtime I/O messages issued by f90

TABLE A-2 f90 Runtime I/O Messages

Error	Message
1000	format error
1001	illegal unit number
1002	formatted I/O on unformatted unit
1003	unformatted I/O on formatted unit
1004	direct-access I/O on sequential-access unit
1005	sequential-access I/O on direct-access unit
1006	device does not support BACKSPACE
1007	off beginning of record
1008	can't stat file
1009	no * after repeat count
1010	record too long
1011	truncation failed
1012	incomprehensible list input

TABLE A-2 f90 Runtime I/O Messages (continued)

Error	Message
1013	out of free space
1014	unit not connected
1015	read unexpected character
1016	illegal logical input field
1017	'new' file exists
1018	can't find 'old' file
1019	unknown system error
1020	requires seek ability
1021	illegal argument
1022	negative repeat count
1023	illegal operation for channel or device
1024	reentrant I/O
1025	incompatible specifiers in open
1026	illegal input for namelist
1027	error in FILEOPT parameter
1028	writing not allowed
1029	reading not allowed
1030	integer overflow on input
1031	floating-point overflow on input
1032	floating-point underflow on input
1051	default input unit closed
1052	default output unit closed
1053	direct-access READ from unconnected unit
1054	direct-access WRITE to unconnected unit
1055	unassociated internal unit
1056	null reference to internal unit
1057	empty internal file
1058	list-directed I/O on unformatted unit
1059	namelist I/O on unformatted unit
1060	tried to write past end of internal file

TABLE A-2 f90 Runtime I/O Messages *(continued)*

Error	Message
1061	unassociated ADVANCE specifier
1062	ADVANCE specifier is not 'YES' or 'NO'
1063	EOR specifier present for advancing input
1064	SIZE specifier present for advancing input
1065	negative or zero record number
1066	record not in file
1067	corrupted format
1068	unassociated input variable
1069	more I/O-list items than data edit descriptors
1070	zero stride in subscript triplet
1071	zero step in implied DO-loop
1072	negative field width
1073	zero-width field
1074	character string edit descriptor reached on input
1075	Hollerith edit descriptor reached on input
1076	no digits found in digit string
1077	no digits found in exponent
1078	scale factor out of range
1079	digit equals or exceeds radix
1080	unexpected character in integer field
1081	unexpected character in real field
1082	unexpected character in logical field
1083	unexpected character in integer value
1084	unexpected character in real value
1085	unexpected character in complex value
1086	unexpected character in logical value
1087	unexpected character in character value
1088	unexpected character before NAMELIST group name
1089	NAMELIST group name does not match the name in the program
1090	unexpected character in NAMELIST item

TABLE A-2 f90 Runtime I/O Messages (continued)

Error	Message
1091	unmatched parenthesis in NAMELIST item name
1092	variable not in NAMELIST group
1093	too many subscripts in NAMELIST object name
1094	not enough subscripts in NAMELIST object name
1095	zero stride in NAMELIST object name
1096	empty section subscript in NAMELIST object name
1097	subscript out of bounds in NAMELIST object name
1098	empty substring in NAMELIST object name
1099	substring out of range in NAMELIST object name
1100	unexpected component name in NAMELIST object name
1111	unassociated ACCESS specifier
1112	unassociated ACTION specifier
1113	unassociated BINARY specifier
1114	unassociated BLANK specifier
1115	unassociated DELIM specifier
1116	unassociated DIRECT specifier
1117	unassociated FILE specifier
1118	unassociated FMT specifier
1119	unassociated FORM specifier
1120	unassociated FORMATTED specifier
1121	unassociated NAME specifier
1122	unassociated PAD specifier
1123	unassociated POSITION specifier
1124	unassociated READ specifier
1125	unassociated READWRITE specifier
1126	unassociated SEQUENTIAL specifier
1127	unassociated STATUS specifier
1128	unassociated UNFORMATTED specifier
1129	unassociated WRITE specifier
1130	zero length file name

TABLE A-2 f90 Runtime I/O Messages *(continued)*

Error	Message
1131	ACCESS specifier is not 'SEQUENTIAL' or 'DIRECT'
1132	ACTION specifier is not 'READ', 'WRITE' or 'READWRITE'
1133	BLANK specifier is not 'ZERO' or 'NULL'
1134	DELIM specifier is not 'APOSTROPHE', 'QUOTE', or 'NONE'
1135	unexpected FORM specifier
1136	PAD specifier is not 'YES' or 'NO'
1137	POSITION specifier is not 'APPEND', 'ASIS', or 'REWIND'
1138	RECL specifier is zero or negative
1139	no record length specified for direct-access file
1140	unexpected STATUS specifier
1141	status is specified and not 'OLD' for connected unit
1142	STATUS specifier is not 'KEEP' or 'DELETE'
1143	status 'KEEP' specified for a scratch file
1144	impossible status value
1145	a file name has been specified for a scratch file
1146	attempting to open a unit that is being read from or written to
1147	attempting to close a unit that is being read from or written to
1148	attempting to open a directory
1149	status is 'OLD' and the file is a dangling symbolic link
1150	status is 'NEW' and the file is a symbolic link
1151	no free scratch file names
1161	device does not support REWIND
1162	read permission required for BACKSPACE
1163	BACKSPACE on direct-access unit
1164	BACKSPACE on binary unit
1165	end-of-file seen while backspacing
1166	write permission required for ENDFILE
1167	ENDFILE on direct-access unit
1181	attempting to allocate an allocated array
1182	deallocating an unassociated pointer

TABLE A-2 f90 Runtime I/O Messages *(continued)*

Error	Message
1183	deallocating an unallocated allocatable array
1184	deallocating an allocatable array through a pointer
1185	deallocating an object not allocated by an ALLOCATE statement
1186	deallocating a part of an object
1187	deallocating a larger object than was allocated
1191	unallocated array passed to array intrinsic function
1192	illegal rank
1193	small source size
1194	zero array size
1195	negative elements in shape
1196	illegal kind
1197	nonconformable array
2001	invalid constant, structure, or component name
2002	handle not created
2003	character argument too short
2004	array argument too long or too short
2005	end of file, record, or directory stream

Features Release History

This Appendix lists the new and changed features in this and previous release of `f77` and `f90`:

`f77` New Features and Changes

This section lists the new features and behavior changes specific to `f77` in this and previous releases.

Features in `f77` 5.0 That are New Since 4.2

`f77` 5.0 includes the following new and changed features:

- New options:
 - `-fpovert` detects floating-point overflows in I/O processing.
 - `-xcode=code` specifies the memory address model on SPARC platforms.
 - `-xcommonchk` enables runtime checking for inconsistent COMMON block declarations.
 - `-xmaxopt` enables the `OPT=n` pragma and controls the maximum optimization level allowed by `OPT` pragmas in the source code.
 - `-xprefetch` allows the compiler to generate prefetch instructions on UltraSPARC II platforms.
 - `-xvector` allows the compiler to replace certain math library calls within DO loops with single calls to a vectorized math routine.
- Changed options:

- `-xcrossfile[=n]` – optional level number added.
 - `-fns[={yes|no}]` – optional yes/no added.
 - `-Ztha` – option now ignored.
- New Features:
- Compile for the 64-bit Solaris 7 environment on 64-bit SPARC platforms with `-xarch=v9` or `v9a`.
 - Support in the I/O library for large files (larger than 2 Gigabytes).
 - Support for large arrays on 64-bit Solaris 7 environments.
 - Dynamic arrays (local arrays with dynamic size) implemented (see *FORTRAN 77 Language Reference Manual*).
 - The `REDUCTION` directive accepts arrays in the list of variables.
 - *SPARC*: A `TASKCOMMON` directive declares variables in `COMMON` to be private.
 - Fortran 90 style constants that allows specification of byte size (for example, `12345678_8` for a 64-bit, 8-byte, constant).
 - New optimization pragma allows setting the compilers optimization level on a routine by routine basis.
 - Year 2000 safe `date_and_time()` library routine.

Features in § 77 4.2 That are New Since 4.0

§ 77 4.2 includes the following features that are new or changed since the 4.0 release:

- New options:
 - `-dbl_align_all`
 - `-errtags=yes|no` and `-erroff=taglist`
 - `-stop_status=no|yes`
 - `-xcrossfile`
 - `-xlic_lib=libs`
 - `-xpp=fpp|cpp`
 - `-xtypemap=type:spec.`
- Changed options:
 - Options
 - `-fround`, `-fsimple`, `-ftrap`, `-xprofile=tcov`, `-xspace`, `-xunroll` now available on Intel platforms.
 - `-xtarget`, `-xarch`, `-xchip` expanded for SPARC Ultra and Intel platforms.
 - `-vax=` expanded to enable selection/deselection of individual VAX/VMS Fortran features.
 - Default sourcefile preprocessor is `fpp(1)` rather than `cpp(1)`.

Features in f77 4.0 that are New Since 3.0/3.0.1

f77 4.0 includes the following features that are new or changed since 3.0/3.0.1:

- The `DO SERIAL` and `DO SERIAL*` parallel directives have been added, and the `DO ALL` directive expanded.
- A directive for unrolling loops has been added.
- The `-I dir` option now also affects the `f77 INCLUDE` statement, not only the preprocessor `#include` directive.
- The Incremental Linker is available. It provides faster linking and speeds up development.
- The `-oldstruct` command-line option has been deleted.
- The following new synonyms have been added: `-xautopar`, `-xdepend`, `-xexplicitpar`, `-xloopinfo`, `-xparallel`, `-xreduction`, and `-xvpara`.
- The `-stackvar` restrictions `EQUIVALANCE`, `NAMELIST`, `STRUCTURE`, and `RECORD` have been removed.
- New options have been added (and some changed):

TABLE B-1 New Features in f77 4.0 Since 3.0/3.0.1

<code>-arg=local</code>	Pass by value result.
<code>-copyargs</code>	Allow assignment to constant arguments.
<code>-dbl</code>	Double the default size for integers, reals, and so forth.
<code>-ext_names=e</code>	Make external names with or without underscores.
<code>-fns</code>	Turn on SPARC non-standard floating-point mode (<i>SPARC</i>).
<code>-fround=r</code>	Set the IEEE rounding mode in effect at startup (<i>SPARC</i>).
<code>-fsimple[=n]</code>	Allow levels of simple floating-point model.
<code>-ftrap=t</code>	Set the IEEE trapping mode in effect at startup (<i>SPARC</i>).
<code>-mp=x</code>	Use either Sun-style or Cray-style MP directives (<i>SPARC</i>).
<code>-O5</code>	Attempt the highest level of optimization.
<code>-pad=p</code>	Pad local variables or common blocks

TABLE B-1 New Features in f77 4.0 Since 3.0/3.0.1 (continued)

<code>-vax=v</code>	Specify a choice of VMS features to use.
<code>-xarch=a</code>	Limit the set of instructions the compiler may use (SPARC).
<code>-xcache=c</code>	Define the cache properties for use by the optimizer (SPARC).
<code>-xchip=c</code>	Specify the target processor for use by the optimizer (SPARC).
<code>-xhelp=h</code>	Show help information for README file or for options (flags).
<code>-xildoff</code>	Turn off the Incremental Linker (SPARC).
<code>-xildon</code>	Turn on the Incremental Linker (SPARC).
<code>-xprofile=p</code>	Collect data for a profile or use a profile to optimize (SPARC).
<code>-xregs=r</code>	Specify the usage of registers for the generated code (SPARC).
<code>-xsafe=mem</code>	Allow compiler to assume no memory-based traps (SPARC).
<code>-xspace</code>	Do no optimizations that increase the code size (SPARC).
<code>-xtarget=t</code>	Specify target system for instruction set (SPARC).
<code>-ztext</code>	Do not make the library if relocations remain.

- DO-loop code is now implemented differently to allow better optimization and loop parallelization. Legal DO-loops behave exactly the same as before; however, illegal DO-loops—zero-step, loop variable modified within the loop—may display different behavior.
- Full 64-bit integers have been added. With `-dbl`, integers not declared with a specified size are turned into full 64-bit integers.
- The following `libV77` library routines: `date`, `mvbits`, `ran`, and `secnds`, are now folded into the `libF77` library. That is, you no longer need to compile with the `-lV77` option to get these routines.
- The `OPEN` statement now contains a new keyword specifier, `ACTION=act`, where `act` is `READ`, `WRITE`, or `READWRITE`.

FORTRAN 77 Upward Compatibility

The FORTRAN 77 5.0 *source* is compatible with earlier releases, except for minor changes due to operating system changes and bug fixes.

Fortran 3.0/3.0.1 to 4.0

Executables (`a.out`), libraries (`.a`), and object files (`.o`) compiled and linked in Fortran 3.0/3.0.1 under Solaris 2 are compatible with Fortran 5.0 under Solaris 2.

BCP: Running Applications from Solaris 1 in 2

You must install the Binary Compatibility Package for the executable to run.

Executables compiled and linked in Solaris 1 do run in Solaris 2, but they do not run as fast as when they are compiled and linked under the appropriate Solaris release.

Libraries (`.a`) and object files (`.o`) compiled and linked in Fortran 2.0.1 under Solaris 1 are *not* compatible with Fortran 5.0.

£90 New Features and Changes

This section lists the new features and behavior changes specific to this 5.0 release of £90.

New Features in £90 2.0 Since 1.2:

- New options:
 - Most £77 options now recognized by £90.
 - `-fpovert` detects floating-point overflows in I/O processing.
 - `-xcode=code` specifies the memory address model on SPARC platforms.
 - `-xcommonchk` enables runtime checking for inconsistent COMMON block declarations.
 - `-xprefetch` allows the compiler to generate prefetch instructions on UltraSPARC II platforms.

- `-xvector` allows the compiler to replace certain math library calls within DO loops with single calls to a vectorized math routine.
- Changed options:
 - `-xcrossfile[=n]` - optional level number added.
 - `-fns[={yes|no}]` - optional yes/no added.
 - `-ztha` - option now ignored.
- New Features:
 - Compile for the 64-bit Solaris 7 environment on 64-bit SPARC platforms with `-xarch=v9` or `v9a`.
 - Support in the I/O library for large files (larger than 2 Gigabytes).
 - Support for large arrays on 64-bit Solaris 7 environments.
 - Accepts Sun-style directives by default.
 - The `REDUCTION` directive accepts arrays in the list of variables.
 - *SPARC*: A `TASKCOMMON` directive declares variables in `COMMON` to be private.
 - New optimization pragma allows setting the compilers optimization level on a routine by routine basis.
- I/O Differences:
 - NAMELIST Output Format:
 - 1.2*: All variables in a single print statement written to a single line without line breaks. *2.0*: Each variable printed to a separate line.
 - 1.2*: Comma used to separate values. *2.0*: Single blank separates values.
 - 1.2*: Repeated values output using the `r*` form: `3*8.22` *2.0*: All repeated values output explicitly: `8.22 8.22 8.22`
 - 1.2*: No trailing zero printing integer floating point: `1.` *2.0*: Floating point integers print with trailing zero: `1.0`
 - 1.2*: Value printed may not be the same value when read into a variable with the same type: `0.1` when read in will print as `0.100000001` *2.0*: Prints the minimum number of digits required to ensure that a value written produces the same value when read back in: `0.1` prints as `0.1`
 - 1.2*: As required by the standard, zero value prints in exponent form. But *1.2* prints `0.E+0` *2.0*: Prints zero as `0.0E+0`
 - 1.2*: Prints a space between the comma and the imaginary part of a complex value: `(1., 0.E+0)` *2.0*: No comma: `(1.0,0.0E+0)`
 - NAMELIST Input Format:

2.0: Allow the group name to be preceded by \$ or & on input. The & is the only form accepted by the Fortran 90 standard, and is what is written by NAMELIST output.

2.0: Accepts \$ as the symbol terminating input except if the last data item in the group is CHARACTER, in which case it is treated as input data.

2.0: Allows NAMELIST input to start in the first column of a record.

- PRINT * no longer comma-delimits output.
- OPEN FORM='BINARY' permits I/O of non-standard raw text without record marks: Opening a file with FORM='BINARY' has roughly the same effect as FORM='UNFORMATTED', except that no record lengths are embedded in the file. Without this data, there is no way to tell where one record begins, or ends. Thus, it is impossible to BACKSPACE a FORM='BINARY' file, because there is no way of telling where to backspace to. A READ on a 'BINARY' file will read as much data as needed to fill the variables on the input list.
- Recursive I/O possible on different units (this is because the f90 I/O library is "MT-Warm").
- RECL=2147483646 ($2^{31}-2$) is the default record length on sequential formatted, list directed, and namelist output. (Default was 267).
- ENCODE and DECODE are recognized and implemented as described in the *FORTRAN 77 Language Reference Manual*.
- Naming of scratch files is the same as with f77.
- Non-advancing I/O is enabled with ADVANCE='NO', as in:

```
write(*,'(a)',ADVANCE='NO') 'n= '      read(*,*) n
```
- Handling of I/O on internal files follows the Fortran 90 standard more closely than was the case with f90 1.2. Also, calls to routines that do internal I/O are allowed on I/O lists. This was not allowed with 1.2 (or f77).
- Operational Differences:
 - Modules are handled differently: Compiling a source code that contains one or more MODULE units now causes an information file (name.mod)to be generated for each module. The name of this information file is the name of the module, in lower case, with .mod suffix. A .mod file must be available before the module can appear on a USE statement. This means that all MODULE files must be compiled (and the module information files created) before compiling any file referencing a MODULE in a USE statement
 - -ftrap=common is the default trapping mode.

- Routines from the Sun Performance Library are automatically linked to perform array operations.
- New Language Elements:
 - Some Fortran 95 elements are implemented: The attributes `PURE` and `ELEMENTAL`. The enhanced forms of `MAXVAL` and `MINVAL`.
 - New data types are recognized: `COMPLEX*32` `REAL*16` `INTEGER*8` (also `*1`, `*2`) `LOGICAL*8` (also `*1`, `*2`)
 - Some data representations have changed from f90 1.2: `INTEGER*2` is now 2 bytes, not 4. `INTEGER*1` is now 1 byte, not 4. `LOGICAL*2` is now 2 bytes, not 4. `LOGICAL*1` is now 1 byte, not 4. This will affect programs that read binary data files containing these data items that were written with f90 programs compiled with the 1.2 compiler. A workaround would be to change the declarations to be `INTEGER*4` or `LOGICAL*4` instead of `*1` or `*2` when compiling with 2.0.
 - Call by value, `%VAL`, is implemented in the same manner as f77. The only difference is that f90 2.0 allows `REAL*8` and `REAL*16` to be passed to C routines as doubles and long doubles.
- f77 and C Interoperability with f90 2.0:
 - To mix f77 and f90 object binaries, link with the f77 compatibility library, `libf77compat`, and not with `libF77`. For example, perform the link step with f90 `..files.. -lf77compat` even if the main program is an f77 program.
 - The structure of f90 `COMMON` is now compatible with f77.
 - f90 *scalar* pointers are compatible with C pointers.

Fortran 90 Features and Differences

This appendix shows some of the major features differences between:

- Standard Fortran 90 and Sun Fortran 90
- FORTRAN 77 and Fortran 90

Features

Sun Fortran 90 provides the following features.

Tabs in the Source

£90 allows the tab character in fixed-form source and in free-form source. Standard Fortran 90 does not allow tabs.

The tab character is not converted to a blank, so the visual placement of tabbed statements depends on the utility you use to edit or display text.

Fixed-Form Source

- For a tab in column one:
 - If the next character is a nonzero digit, then the current line is a *continuation* line; otherwise, the current line is an *initial* line.
- A tab cannot precede a statement label.
- A tab after column one is treated by £90 the same as a blank character, except in literal strings.

Free-Form Source

f90 treats a tab and a blank character as equivalent, except in literal strings.

Continuation Line Limits

f90 and f77 allow 99 continuation lines (1 initial and 98 continuation lines). Standard Fortran 90 allows 19 for fixed-form and 39 for free-form.

Fixed-Form Source Lines

In fixed-form source, lines can be longer than 72 characters. Columns 73 through 96 are ignored. Standard Fortran 90 allows 72-character lines.

Directives

f90 allows directive lines starting with CDIR\$, !DIR\$, CMIC\$, or !MIC\$. They look like comments but are not. For full details on directives, see “Directives” on page C-16. Standard Fortran 90 has no directives.

Tab Form

The tab form source text is defined as follows:

- A tab in any of columns 1 through 6 makes the line as a tab form source line.
- A comment indicator or a statement number may precede the tab.
- If a tab is the first nonblank character, then:
 - If the character after the tab is anything other than a nonzero digit, then the text following the tab is an initial line.
 - If there is a nonzero digit after the first tab, the line is a continuation line. The text following the nonzero digit is the next part of the statement.
- The default maximum line length is 72 columns for fixed form and 132 for free form.

Example: The tab form source on the left is treated as shown on the right.

```
!^IUses of tabs
^I CHARACTER *3 A = 'A'
^I INTEGER B = 2
```

```

^IREAL C = 3.0
^IWRITE(*,9) A, B, C
9^IFORMAT(1X, A3,
^I1 I3,
^I2 F9.1 )
^IEND
!      Uses of tabs
      CHARACTER *3 A = 'A'
      INTEGER B = 2
      REAL C = 3.0
      WRITE(*,9) A, B, C
9      FORMAT(1X, A3,
1 I3,
2 F9.1 )
      END

```

In the example above, “^I” is a way of indicating the tab character, and the line starting with “1” and “2” are continuation lines. The coding is shown to illustrate various tab situations, and not to advocate any one style.

Source Form Assumed

The source form assumed by f90 depends on options, directives, and suffixes.

TABLE C-1 F90 Source Form Command-line options

Option	Action
-fixed	Interpret all source files as Fortran <i>fixed</i> form
-free	Interpret all source files as Fortran <i>free</i> form

If the `-free` or `-fixed` option is used, that overrides the file name suffix.

TABLE C-2 F90 File name suffixes

Suffix	Source Form
.f90	Fortran <i>free-form</i> source files
.f	Fortran <i>fixed-form</i> source files or ANSI standard FORTRAN 77 source files
.for	Same as .f.
.ftn	Same as .f.
other	None—file name is passed to the linker

If either a `FREE` or `FIXED` directive is used, that overrides the option and file name suffix.

Mixing Forms

Some mixing of source forms is allowed.

- In the same `f90` command, some source files can be fixed form, some free.
- In the same file, free form *can* be mixed with fixed form by using directives.
- In the same program unit, tab form *can* be mixed with free or fixed form.

Case

Sun Fortran 90 is case insensitive. That means that a variable `AbcDeF` is treated as if it were spelled `abcdef`, or `abcdeF`, etc. “Compatibility with FORTRAN 77” on page C-19

Boolean Type

`f90` supports constants and expressions of Boolean type. There are no Boolean variables or arrays, and there is no Boolean type statement.

Miscellaneous Rules Governing Boolean Type

- *Masking*-A bitwise logical expression has a Boolean result; each of its bits is the result of one or more logical operations on the corresponding bits of the operands.
- For binary arithmetic operators, and for relational operators:
 - If one operand is Boolean, the operation is performed with no conversion.
 - If both operands are Boolean, the operation is performed as if they were integers.
- No user-specified function can generate a Boolean result, although some (nonstandard) intrinsics can.
- Boolean and logical types differ as follows:
 - Variables, arrays, and functions can be of logical type, but they cannot be Boolean type.
 - There is a LOGICAL statement, but no BOOLEAN statement.
 - A logical variable or constant represents only one value. A Boolean constant can represent as many as 32 values.
 - A logical expression yields one value. A Boolean expression can yield as many as 32 values.
 - Logical entities are invalid in arithmetic, relational, or bitwise logical expressions. Boolean entities are valid in all three.

Alternate Forms of Boolean Constants

Fortran 90 allows a Boolean constant (octal, hexadecimal, or Hollerith) in the following alternate forms (no binary). Variables cannot be declared Boolean. Standard Fortran does not allow these forms.

Octal

dddddB, where *d* is any octal digit

- You can use the letter B or b.
- There can be 1 to 11 octal digits (0 through 7).
- 11 octal digits represent a full 32-bit word, with the leftmost digit allowed to be 0, 1, 2, or 3.
- Each octal digit specifies three bit values.

- The last (rightmost) digit specifies the content of the rightmost three bit positions (bits 29, 30, and 31).
- If less than 11 digits are present, the value is right-justified—it represents the rightmost bits of a word: bits *n* through 31. The other bits are 0.
- Blanks are ignored.

Within an I/O format specification, the letter **B** indicates *binary* digits; elsewhere it indicates *octal* digits.

Hexadecimal

x'ddd' or *x"ddd"*, where *d* is any hexadecimal digit

- There can be 1 to 8 hexadecimal digits (0 through 9, A-F).
- Any of the letters can be uppercase or lowercase (X, x, A-F, a-f).
- The digits must be enclosed in either apostrophes or quotes.
- Blanks are ignored.
- The hexadecimal digits may be preceded by a + or - sign.
- 8 hexadecimal digits represent a full 32-bit word and the binary equivalents correspond to the contents of each bit position in the 32-bit word.
- If less than 8 digits are present, the value is right-justified—it represents the rightmost bits of a word: bits *n* through 31. The other bits are 0.

Hollerith

Accepted forms for Hollerith data are:

<i>n</i> H...	'... 'H	"... "H
<i>n</i> L...	'... 'L	"... "L
<i>n</i> R...	'... 'R	"... "R

Above, “...” is a string of characters and *n* is the character count.

- A Hollerith constant is type Boolean.
- If any character constant is in a bitwise logical expression, the expression is evaluated as Hollerith.
- A Hollerith constant can have 1 to 4 characters.

Examples: Octal and hexadecimal constants.

Boolean Constant	Internal Octal for 32-bit word
0B	0000000000
77740B	0000077740
X"ABE"	0000005276
X"-340"	3777776300
X"1 2 3"	0000000443
X"FFFFFFFFFFFFFF"	3777777777

Examples: Octal and hexadecimal in assignment statements.

i = 1357B
j = X"28FF"
k = X"-5A"

Use of an octal or hexadecimal constant in an arithmetic expression can produce undefined results and do not generate syntax errors.

Alternate Contexts of Boolean Constants

£90 allows BOZ constants in the places other than DATA statements.

B' <i>bbb</i> '	O' <i>ooo</i> '	Z' <i>zzz</i> '
B" <i>bbb</i> "	O" <i>ooo</i> "	Z" <i>zzz</i> "

If these are assigned to a real variable, no type conversion occurs.

Standard Fortran allows these only in DATA statements.

Abbreviated Size Notation for Numeric Data Types

£90 allows the following nonstandard type declaration forms in declaration statements, function statements, and IMPLICIT statements.

TABLE C-3 Size Notation for Numeric Data Types

Nonstandard	Declarator	Short Form	Meaning
INTEGER*1	INTEGER(KIND=1)	INTEGER(1)	One-byte signed integers
INTEGER*2	INTEGER(KIND=2)	INTEGER(2)	Two-byte signed integers
INTEGER*4	INTEGER(KIND=4)	INTEGER(4)	Four-byte signed integers
LOGICAL*1	LOGICAL(KIND=1)	LOGICAL(1)	One-byte logicals
LOGICAL*2	LOGICAL(KIND=2)	LOGICAL(2)	Two-byte logicals
LOGICAL*4	LOGICAL(KIND=4)	LOGICAL(4)	Four-byte logicals
REAL*4	REAL(KIND=4)	REAL(4)	IEEE single-precision floating-point (Four-byte)
REAL*8	REAL(KIND=8)	REAL(8)	IEEE double-precision floating-point (Eight-byte)
COMPLEX*8	COMPLEX(KIND=4)	COMPLEX(4)	Single-precision complex (Four-bytes each part)
COMPLEX*16	COMPLEX(KIND=8)	COMPLEX(8)	Double-precision complex (Eight-bytes each part)

The form in column one is nonstandard Fortran 90, though in common use. The kind numbers in column two can vary by vendor.

Cray Pointers

A *Cray pointer* is a variable whose value is the address of another entity, which is called the *pointee*.

f90 supports Cray pointers; Standard Fortran 90 does not.

Syntax

The Cray `POINTER` statement has the following format:

```
POINTER ( pointer_name, pointee_name [array_spec] ), ...
```

Where *pointer_name*, *pointee_name*, and *array_spec* are as follows:

<i>pointer_name</i>	Pointer to the corresponding <i>pointee_name</i> . <i>pointer_name</i> contains the address of <i>pointee_name</i> . Must be: a scalar variable name (but not a structure) Cannot be: a constant, a name of a structure, an array, or a function
<i>pointee_name</i>	Pointee of the corresponding <i>pointer_name</i> Must be: a variable name, array declarator, or array name
<i>array_spec</i>	If <i>array_spec</i> is present, it must be explicit shape, (constant or nonconstant bounds), or assumed-size.

Example: Declare Cray pointers to two pointees.

```
POINTER ( p, b ), ( q, c )
```

The above example declares Cray pointer *p* and its pointee *b*, and Cray pointer *q* and its pointee *c*.

Example: Declare a Cray pointer to an array.

```
POINTER ( ix, x(n, 0:m) )
```

The above example declares Cray pointer *ix* and its pointee *x*; and declares *x* to be an array of dimensions *n* by *m*-1.

Purpose of Cray Pointers

You can use pointers to access user-managed storage by dynamically associating variables to particular locations in a block of storage.

Cray pointers allow accessing absolute memory locations.

Cray pointers do not provide convenient manipulation of linked lists because (for optimization purposes) it is assumed that no two pointers have the same value.

Cray Pointers and Fortran Pointers

Cray pointers are declared as follows:

```
POINTER ( pointer_name, pointee_name [array_spec] )
```

Fortran pointers are declared as follows:

```
POINTER object_name
```

The two kinds of pointers cannot be mixed.

Features of Cray Pointers

- Whenever the pointee is referenced, `£90` uses the current value of the pointer as the address of the pointee.
- The Cray pointer type statement declares both the pointer and the pointee.
- The Cray pointer is of type Cray pointer.
- The value of a Cray pointer occupies one storage unit. Its range of values depends on the size of memory for the machine in use.
- The Cray pointer can appear in a `COMMON` list or as a dummy argument.
- The Cray pointee has no address until the value of the Cray pointer is defined.
- If an array is named as a pointee, it is called a *pointee array*.

Its array declarator can appear in:

- A separate type statement
 - A separate `DIMENSION` statement
 - The pointer statement itself
- If the array declarator is in a subprogram, the dimensioning can refer to:
 - Variables in a common block, or
 - Variables that are dummy arguments
 - The size of each dimension is evaluated on entrance to the subprogram, not when the pointee is referenced.

Restrictions on Cray Pointers

- If *pointee_name* is of character type, it must be a variable typed `CHARACTER*(*)`.
- If *pointee_name* is an array declarator, it must be explicit shape, (constant or nonconstant bounds), or assumed-size.
- An array of Cray pointers is not allowed.

- A Cray pointer cannot be:
 - Pointed to by another Cray pointer or by a Fortran pointer.
 - A component of a structure.
 - Declared to be any other data type.
- A Cray pointer cannot appear in:
 - A `PARAMETER` statement or in a type declaration statement that includes the `PARAMETER` attribute.
 - A `DATA` statement.

Restrictions on Cray Pointees

- A Cray pointee cannot appear in a `SAVE`, `DATA`, `EQUIVALENCE`, `COMMON`, or `PARAMETER` statement.
- A Cray pointee cannot be a dummy argument.
- A Cray pointee cannot be a function value.
- A Cray pointee cannot be a structure or a structure component.
- A Cray pointee cannot be of a derived type.

Note - Cray pointees can be of type character, but their Cray pointers are different from other Cray pointers. The two kinds cannot be mixed in the same expression.

Usage of Cray Pointers

Cray pointers can be assigned values as follows:

- Set to an absolute address
 - Example: `q = 0`
- Assigned to or from integer variables, plus or minus expressions
 - Example: `p = q + 100`
- Cray pointers are not integers. You cannot assign them to a real variable.
- The `LOC` function (nonstandard) can be used to define a Cray pointer.
 - Example: `p = LOC(x)`

Example: Use Cray pointers as described above.

```
SUBROUTINE sub ( n )
COMMON pool(100000)
INTEGER blk(128), word64
REAL a(1000), b(n), c(100000-n-1000)
```

```

    POINTER ( pblk, blk ), ( ia, a ), ( ib, b ), &
      ( ic, c ), ( address, word64 )
    DATA address / 64 /
    pblk = 0
    ia = LOC( pool )
    ib = ia + 1000
    ic = ib + n
    ...

```

Remarks about the above example:

- word64 refers to the contents of absolute address 64
- blk is an array that occupies the first 128 words of memory
- a is an array of length 1000 located in blank common
- b follows a and is of length n
- c follows b
- a, b, and c are associated with pool
- word64 is the same as blk(17) because Cray pointers are byte address and the integer elements of blk are each 4 bytes long

Optimization and Cray Pointers

For purposes of optimization, f90 assumes the storage of a pointee is never overlaid on the storage of another variable—it assumes that a pointee is not associated with another variable.

Such association could occur in either of two ways:

- A Cray pointer has two pointees, or
- Two Cray pointers are given the same value

Note - The programmer is responsible for preventing such association.

These kinds of association are sometimes done deliberately, such as for equivalencing arrays, but then results can differ depending on whether optimization is turned on or off.

Example: b and c have the same pointer.

```

    POINTER ( p, b ), ( p, c )
    REAL x, b, c
    p = LOC( x )
    b = 1.0
    c = 2.0
    PRINT *, b
    ...

```

Above, because `b` and `c` have the same pointer, assigning 2.0 to `c` gives the same value to `b`. Therefore `b` prints out as 2.0, even though it was assigned 1.0.

Cray Character Pointers

If a pointee is declared as a character type, its Cray pointer is a Cray character pointer.

Purpose of Cray Character Pointers

A Cray character pointer is a special data type that allows `FORTRAN 90` to maintain character strings by keeping track of the following:

- Byte address of the first character of the string
- Length
- Offset

An assignment to a Cray character pointer alters all three. That is, when you change what it points to, all three change.

Declaration of Cray Character Pointers

For a pointee that has been declared with an assumed length character type, the Cray pointer declaration statement declares the pointer to be a Cray character pointer.

1. **Before the Cray pointer declaration statement, declare the pointee as a character type with an assumed length.**
2. **Declare a Cray pointer to that pointee.**
3. **Assign a value to the Cray character pointer.**

You can use functions `CLOC` or `FCD`, both nonstandard intrinsics.

Example: Declare `Ccp` to be a Cray character pointer and use `CLOC` to make it point to character string `s`.

```
CHARACTER*(*) a
POINTER ( Ccp, a )
CHARACTER*80 :: s = "abcdefghijklmnopqrsxyz"
Ccp = CLOC( s )
```

Operations on Cray Character Pointers

You can do the following operations with Cray character pointers:

$Ccp1 + i$
 $Ccp1 - i$
 $i + Ccp1$
 $Ccp1 = Ccp2$
 $Ccp1 \textit{ relational_operator } Ccp2$

where $Ccp1$ and $Ccp2$ are Cray character pointers and i is an integer.

Restrictions on Cray Character Pointers and Pointees

All restrictions to Cray pointers also apply to Cray character pointers. In addition, the following apply:

- A Cray character pointee cannot be an array.
- In a relational operation, a Cray character pointer can be mixed with only another Cray character pointer—not with a Cray pointer, not with an integer.
- A relational operation applies only to the character address and the bit offset; the length field is not involved.
- Cray character pointers must not appear in `EQUIVALENCE` statements, or any storage association statements. (The size can vary with the platform.)
- Cray character pointers are not optimized.
- Code containing Cray character pointers is not parallelized.
- A Cray character pointer in a list of an I/O statement is treated as an integer.

Intrinsics

£90 supports some intrinsic procedures which are extensions beyond the standard.

TABLE C-4 Nonstandard Ininsics

Name	Definition	Type			Notes
		Function	Arguments	Arguments	
CLOC	Get Fortran character descriptor (FCD)	Cray character pointer	character	((C=]c)	NP, I
COT	Cotangent	real	real	((X=]x)	P, E
DDIM	Positive difference	double precision	double precision	((X=]x,[Y=]y)	P, E
FCD	Create Cray character pointer in Fortran character descriptor (FCD) format	Cray pointer	i: integer or Cray pointer j: integer	(([I=]i, [J=]j) i: word address of first character j: character length	NP, I
LEADZ	Get the number of leading 0 bits	integer	Boolean, integer, real, or pointer	((I=]i)	NP, I
POPCNT	Get the number of set bits	integer	Boolean, integer, real, or pointer	((I=]i)	NP, I
POPPAR	Calculate bit population parity	integer	Boolean, integer, real, or pointer	((X=]x)	NP, I

Notes on the above table:

P	The name can be passed as an argument.
NP	The name cannot be passed as an argument.
E	External code for the intrinsic is called at run time.
I	f90 generates inline code for the intrinsic procedure.

Directives

A compiler *directive* directs the compiler to do some special action. Directives are also called *pragmas*.

A compiler directive is inserted into the source program as one or more lines of text. Each line looks like a comment, but has additional characters that identify it as more than a comment for this compiler. For most other compilers, it is treated as a comment, so there is some code portability.

Sun-style directives are the default with f90 (and f77). To switch to Cray-style directives, use the `-mp=cray` compiler command-line flag.

Form of General Directive Lines

General directives have the following syntax.

```
!DIR$ d1, d2, ...
```

A general *directive line* is defined as follows.

- A directive line starts with the 5 characters `CDIR$` or `!DIR$`, followed by:
 - A space
 - A directive
- Spaces before, after, or within a directive are ignored.
- Letters of a directive line can be in uppercase, lowercase, or mixed.

The form varies for fixed-form and free-form source as follows.

Fixed-Form Source

- Put `CDIR$` or `!DIR$` in columns 1 through 5.
- Directives are listed in columns 7 and beyond.
- Columns beyond 72 are ignored.
- An *initial* directive line has a blank in column 6.
- A *continuation* directive line has a nonblank in column 6.

Free-Form Source

- Put `!DIR$` followed by a space anywhere in the line. The `!DIR$` characters are the first nonblank characters in the line (actually, non-whitespace).
- Directives are listed after the space.
- An *initial* directive line has a blank, tab, or newline in the position immediately after the `!DIR$`.
- A *continuation* directive line has a character other than a blank, tab, or newline in the position immediately after the `!DIR$`.

Thus, `!DIR$` in columns 1 through 5 works for both free-form source and fixed-form source.

FIXED and FREE Directives

These directives specify the source form of lines following the directive line.

Scope

They apply to the rest of the *file* in which they appear, or until the next `FREE` or `FIXED` directive is encountered.

Uses

- They allow you to switch source forms within a source file.
- They allow you to switch source forms for an `INCLUDE` file. You insert the directive at the start of the `INCLUDE` file. After the `INCLUDE` file has been processed, the source form reverts back to the form being used prior to processing the `INCLUDE` file.

Restrictions

The `FREE/FIXED` directives:

- Each must appear alone on a compiler directive line (not continued).
- Each can appear anywhere in your source code. Other directives must appear within the program unit they affect.

Example: A `FREE` directive.

```
!DIR$ FREE
DO i = 1, n
  a(i) = b(i) * c(i)
END DO
```

Parallelization Directives

A *parallelization* directive is a special comment that directs the compiler to attempt to parallelize the next DO loop. Currently there is only one parallel directive, DOALL.

The DOALL directive tells the compiler to parallelize the next loop it finds, if possible.

Form of Parallelization Directive Lines

Parallel directives have the following syntax.

```
!MIC$ DOALL [general parameters] [scheduling parameter]
```

A *parallelization directive line* is defined as follows.

- A parallel directive starts with the CMIC\$ or !MIC\$, followed by:
 - A space
 - A directive
 - For some directives, one or more parameters
- Spaces before, after, or within a directive are ignored.
- Letters of a parallelization directive line can be in uppercase, lowercase, or mixed.

The form varies for fixed-form and free-form source as follows.

Fixed

- Put CMIC\$ or !MIC\$ in columns 1 through 5.
- Directives are listed in columns 7 and beyond.
- Columns beyond 72 are ignored.
- An *initial* directive line has a blank in column 6.
- A *continuation* directive line has a nonblank in column 6.

Free

- Put !MIC\$ followed by a space anywhere in the line. The !MIC\$ characters are the first nonblank characters in the line (actually, non-whitespace).

- Directives are listed after the space.
- An *initial* directive line has a blank, tab, or newline in the position immediately after the !MIC\$.
- A *continuation* directive line has a character other than a blank, tab, or newline in the position immediately after the !MIC\$.

Thus, !MIC\$ in columns 1 through 5 works for both free and fixed.

Example: Directive with continuation lines (DOALL directive and parameters.)

```
C$PAR DOALL
!MIC$&  SHARED( a, b, c, n )
!MIC$&  PRIVATE( i )
DO i = 1, n
  a(i) = b(i) * c(i)
END DO
```

Example: Same directive and parameters, with *no* continuation lines.

```
C$PAR DOALL SHARED( a, b, c, n ) PRIVATE( i )
DO i = 1, n
  a(i) = b(i) * c(i)
END DO
```

Compatibility with FORTRAN 77

Source

Standard-conforming FORTRAN 77 source code is compatible with Sun Fortran 90. Use of non-standard extensions, such as VMS Fortran features, are not compatible and may not compile with Sun Fortran 90.

Both f77 and f90 treat all source lines as if they were lowercase (except in quoted character strings). However, unlike f77, f90 has no -U option to force the compiler to be sensitive to both upper and lower case. This may present a problem when mixing f77 and f90 compiled routines. Since a routine compiled by f90 will treat CALL XYZ the same as CALL xyz, and treat them both as if they were CALL xyz, care must be taken to rearrange the way these calls are made. A similar situation will exist when trying to define entry points in f90 compiled routines that are differentiated by case. The clue to potential problems would be the need to use -U with f77.

Linking with f77-Compiled Routines

- To mix f77 and f90 object binaries, link with f90 and the f77 compatibility library, `libf77compat`, and not with `libF77`. For example, perform the link step with

```
f90 ..files. -lf77compat
```

even if the main program is an f77 program.

Example: f90 main and f77 subroutine.

```
demo% cat m.f90
CHARACTER*74 :: c = 'This is a test.'
CALL echo1( c )
END
demo$ cat s.f
SUBROUTINE echo1( a )
CHARACTER*74 a
PRINT*, a
RETURN
END
demo% f77 -c -silent s.f
demo% f90 m.f90 s.o -lf77compat
demo% a.out
This is a test.
demo%
```

- The FORTRAN 77 library is generally compatible with f90.

Example: f90 main calls a routine from the FORTRAN 77 library.

```
demo% cat tdttime.f90
REAL e, dtime, t(2)
e = dtime( t )
DO i = 1, 100000
as = as + cos(sqrt(float(i)))
END DO
e = dtime( t )
PRINT *, "elapsed:", e, ", user:", t(1), ", sys:", t(2)
END
demo% f90 tdttime.f90
demo% a.out
elapsed: 0.14 , user: 0.14 , sys: 0.0E+0
demo%
```

See *dttime(3F)*.

I/O

f77 and f90 are generally I/O compatible for binary I/O, since f90 links to the f77 compatibility library.

Such compatibility includes the following two situations:

- In the same program, you can write some records in f90, then read them in f77.
- An f90 program can write a file. Then an f77 program can read it.

The numbers read back in may or may not equal the numbers written out.

- Unformatted

The numbers read back in do equal the numbers written out.

- Floating-point formatted

The numbers read back in can be different from the numbers written out. This is caused by slightly different base conversion routines, or by different conventions for uppercase/lowercase, spaces, plus or minus signs, and so forth.

Examples: 1.0e12, 1.0E12, 1.0E+12

- List-directed

The numbers read back in can be different from the numbers written out. This can be caused by various layout conventions with commas, spaces, zeros, repeat factors, and so forth.

Example: '0.0' as compared to '.0'

Example: '7' as compared to '7'

Example: '3, 4, 5' as compared to '3 4 5'

Example: '3*0' as compared to '0 0 0'

The above results are from: `integer::v(3)=(/0,0,0/); print *,v`

Example: '0.333333343' as compared to '0.333333'

The above results are from `PRINT *, 1.0/3.0`

Intrinsics

The Fortran 90 standard supports the following new intrinsic functions that FORTRAN 77 does not have.

If you use one of these names in your program, you must add an `EXTERNAL` statement to make f90 use your function rather than the intrinsic one.

Fortran 90 intrinsics:

“ADJUSTL,ADJUSTR,ALL,ALLOCATED,ANY,BIT_SIZE,COUNT,CSHIFT,
DIGITS,DOT_PRODUCT,EOSHIFT,EPSILON,EXPONENT,HUGE,KIND,
LBOUND,LEN_TRIM,MATMUL,MAXEXPONENT,MAXLOC,MAXVAL,MERGE,
MINEXPONENT,MINLOC,MINVAL,NEAREST,PACK,PRECISION,PRESENT,
PRODUCT,RADIX,RANGE,REPEAT,RESHAPE,RRSPACING,SCALE,SCAN,
SELECTED_INT_KIND,SELECTED_REAL_KIND,SET_EXPONENT,SHAPE,

SIZE,SPACING,SPREAD,SUM,TINY,TRANSFER,TRANSPOSE,UBOUND,
UNPACK,VERIFY”

Forward Compatibility

Future releases of f90 are intended to be source code compatible with this release.

Module information files generated by this release of f90 are not guaranteed to be compatible with future releases.

Mixing Languages

On Solaris systems, routines written in C can be combined with Fortran programs, since these languages have common calling conventions.

Module Files

Compiling a file containing a Fortran 90 `MODULE` generates a module file (`.mod` file) for every `MODULE` encountered in the source. The file name is derived from the name of the `MODULE`; file `xyz.mod` (all lowercase) will be created for `MODULE xyz`.

By default, such files are usually sought in the current working directory. The `-mdir` option allows you to tell f90 to seek them in an additional location.

The `.mod` files cannot be stored into an archive file, or concatenated into a single file.

-xtarget Platform Expansions

This Appendix details the `-xtarget` option platform system names and their expansions.

Each specific value for `-xtarget` expands into a specific set of values for the `-xarch`, `-xchip`, and `-xcache` options, as shown in the following table. Run `fpversion(1)` to determine the target definitions on any system.

For example:

`-xtarget=sun4/15` means `-xarch=v8a -xchip=micro -xcache=2/16/1`

TABLE D-1 The `-xtarget` Expansions

-xtarget	-xarch	-xchip	-xcache
sun4/15	v8a	micro	2/16/1
sun4/20	v7	old	64/16/1
sun4/25	v7	old	64/32/1
sun4/30	v8a	micro	2/16/1
sun4/40	v7	old	64/16/1
sun4/50	v7	old	64/32/1
sun4/60	v7	old	64/16/1

TABLE D-1 The `-xtarget` Expansions *(continued)*

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
sun4/65	v7	old	64/16/1
sun4/75	v7	old	64/32/1
sun4/110	v7	old	2/16/1
sun4/150	v7	old	2/16/1
sun4/260	v7	old	128/16/1
sun4/280	v7	old	128/16/1
sun4/330	v7	old	128/16/1
sun4/370	v7	old	128/16/1
sun4/390	v7	old	128/16/1
sun4/470	v7	old	128/32/1
sun4/490	v7	old	128/32/1
sun4/630	v7	old	64/32/1
sun4/670	v7	old	64/32/1
sun4/690	v7	old	64/32/1
sselc	v7	old	64/32/1
ssipc	v7	old	64/16/1
ssipx	v7	old	64/32/1
sslc	v8a	micro	2/16/1
sslt	v7	old	64/32/1

TABLE D-1 The `-xtarget` Expansions *(continued)*

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
sslx	v8a	micro	2/16/1
sslx2	v8a	micro2	8/16/1
ssslc	v7	old	64/16/1
ss1	v7	old	64/16/1
sslplus	v7	old	64/16/1
ss2	v7	old	64/32/1
ss2p	v7	powerup	64/32/1
ss4	v8a	micro2	8/16/1
ss4/85	v8a	micro2	8/16/1
ss4/110	v8a	micro2	8/16/1
ss5	v8a	micro2	8/16/1
ss5/85	v8a	micro2	8/16/1
ss5/110	v8a	micro2	8/16/1
ssvyger	v8a	micro2	8/16/1
ss10	v8	super	16/32/4
ss10/hs11	v8	hyper	256/64/1
ss10/hs12	v8	hyper	256/64/1
ss10/hs14	v8	hyper	256/64/1
ss10/20	v8	super	16/32/4

TABLE D-1 The `-xtarget` Expansions (continued)

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
ss10/hs21	v8	hyper	256/64/1
ss10/hs22	v8	hyper	256/64/1
ss10/30	v8	super	16/32/4
ss10/40	v8	super	16/32/4
ss10/41	v8	super	16/32/4:1024/32/1
ss10/50	v8	super	16/32/4
ss10/51	v8	super	16/32/4:1024/32/1
ss10/61	v8	super	16/32/4:1024/32/1
ss10/71	v8	super2	16/32/4:1024/32/1
ss10/402	v8	super	16/32/4
ss10/412	v8	super	16/32/4:1024/32/1
ss10/512	v8	super	16/32/4:1024/32/1
ss10/514	v8	super	16/32/4:1024/32/1
ss10/612	v8	super	16/32/4:1024/32/1
ss10/712	v8	super2	16/32/4:1024/32/1
ss20	v8	super	16/32/4:1024/32/1
ss20/hs11	v8	hyper	256/64/1
ss20/hs12	v8	hyper	256/64/1
ss20/hs14	v8	hyper	256/64/1

TABLE D-1 The `-xtarget` Expansions (continued)

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
ss20/hs21	v8	hyper	256/64/1
ss20/hs22	v8	hyper	256/64/1
ss20/50	v8	super	16/32/4
ss20/51	v8	super	16/32/4:1024/32/1
ss20/61	v8	super	16/32/4:1024/32/1
ss20/71	v8	super2	16/32/4:1024/32/1
ss20/151	v8	hyper	512/64/1
ss20/152	v8	hyper	512/64/1
ss20/502	v8	super	16/32/4
ss20/512	v8	super	16/32/4:1024/32/1
ss20/514	v8	super	16/32/4:1024/32/1
ss20/612	v8	super	16/32/4:1024/32/1
ss20/712	v8	super	16/32/4:1024/32/1
ss600/41	v8	super	16/32/4:1024/32/1
ss600/51	v8	super	16/32/4:1024/32/1
ss600/61	v8	super	16/32/4:1024/32/1
ss600/120	v7	old	64/32/1
ss600/140	v7	old	64/32/1
ss600/412	v8	super	16/32/4:1024/32/1

TABLE D-1 The `-xtarget` Expansions (continued)

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
ss600/512	v8	super	16/32/4:1024/32/1
ss600/514	v8	super	16/32/4:1024/32/1
ss600/612	v8	super	16/32/4:1024/32/1
ss1000	v8	super	16/32/4:1024/32/1
sc2000	v8	super	16/32/4:2048/64/1
cs6400	v8	super	16/32/4:2048/64/1
solb5	v7	old	128/32/1
solb6	v8	super	16/32/4:1024/32/1
ultra	v8	ultra	16/32/1:512/64/1
ultra2	v8	ultra2	16/32/1:512/64/1
ultra2i	v8	ultra2i	16/32/1:512/64/1
ultra1/140	v8	ultra	16/32/1:512/64/1
ultra1/170	v8	ultra	16/32/1:512/64/1
ultra1/200	v8	ultra	16/32/1:512/64/1
ultra2/1170	v8	ultra	16/32/1:512/64/1
ultra2/1200	v8	ultra	16/32/1:1024/64/1
ultra2/1300	v8	ultra2	16/32/1:2048/64/1
ultra2/2170	v8	ultra	16/32/1:512/64/1
ultra2/2200	v8	ultra	16/32/1:1024/64/1

TABLE D-1 The `-xtarget` Expansions *(continued)*

<code>-xtarget</code>	<code>-xarch</code>	<code>-xchip</code>	<code>-xcache</code>
ultra2/2300	v8	ultra2	16/32/1:2048/64/1
entr2	v8	ultra	16/32/1:512/64/1
entr2/1170	v8	ultra	16/32/1:512/64/1
entr2/2170	v8	ultra	16/32/1:512/64/1
entr2/1200	v8	ultra	16/32/1:512/64/1
entr2/2200	v8	ultra	16/32/1:512/64/1
entr150	v8	ultra	16/32/1:512/64/1
entr3000	v8	ultra	16/32/1:512/64/1
entr4000	v8	ultra	16/32/1:512/64/1
entr5000	v8	ultra	16/32/1:512/64/1
entr6000	v8	ultra	16/32/1:512/64/1

For x86: `-xtarget=` accepts:

- generic or native
- 386 (equivalent to `-386` option) or 486 (equivalent to `-486` option)
- pentium (equivalent to `-pentium` option) or `pentium_pro`

Index
