Contents

Preface ...................................................................................................................................................13

1 Introduction .........................................................................................................................................17
  1.1 Standards Conformance ............................................................................................................ 17
  1.2 Features of the Fortran Compiler .............................................................................................. 18
  1.3 Other Fortran Utilities ............................................................................................................... 18
  1.4 Debugging Utilities .................................................................................................................. 19
  1.5 Sun Performance Library ........................................................................................................ 19
  1.6 Interval Arithmetic ..................................................................................................................... 19
  1.7 Man Pages ................................................................................................................................ 19
  1.8 README Files ........................................................................................................................... 20
  1.9 Command-Line Help .................................................................................................................. 21

2 Using Sun Studio Fortran ...................................................................................................................23
  2.1 A Quick Start ............................................................................................................................... 23
  2.2 Invoking the Compiler .............................................................................................................. 25
    2.2.1 Compile-Link Sequence ..................................................................................................... 25
    2.2.2 Command-Line File Name Conventions ........................................................................ 25
    2.2.3 Source Files ......................................................................................................................... 26
    2.2.4 Source File Preprocessors ................................................................................................. 27
    2.2.5 Separate Compiling and Linking ...................................................................................... 27
    2.2.6 Consistent Compiling and Linking .................................................................................. 27
    2.2.7 Unrecognized Command-Line Arguments ....................................................................... 28
    2.2.8 Modules ............................................................................................................................... 28
  2.3 Directives .....................................................................................................................................29
    2.3.1 General Directives ............................................................................................................... 29
    2.3.2 Parallelization Directives .................................................................................................. 36
2.4 Library Interfaces and `system.inc` ................................................................................................................. 36
2.5 Compiler Usage Tips ............................................................................................................................................. 37
   2.5.1 Determining Hardware Platform .............................................................................................................. 37
   2.5.2 Using Environment Variables .................................................................................................................... 38
   2.5.3 Memory Size .................................................................................................................................................. 38

3 Fortran Compiler Options ....................................................................................................................................... 41
   3.1 Command Syntax .............................................................................................................................................. 41
   3.2 Options Syntax .................................................................................................................................................. 41
   3.3 Options Summary .............................................................................................................................................. 43
      3.3.1 Commonly Used Options .............................................................................................................................. 48
      3.3.2 Macro Flags ................................................................................................................................................... 48
      3.3.3 Backward Compatibility and Legacy Options .................................................................................................. 49
      3.3.4 Obsolete Option Flags .................................................................................................................................. 49
   3.4 Options Reference ............................................................................................................................................ 50
      3.4.1 –a .................................................................................................................................................................. 50
      3.4.2 –aligncommon=[1|2|4|8|16] ........................................................................................................................ 51
      3.4.3 –ansi .............................................................................................................................................................. 51
      3.4.4 –arg=local .................................................................................................................................................... 51
      3.4.5 –autopar ....................................................................................................................................................... 52
      3.4.6 –B {static|dynamic} ..................................................................................................................................... 52
      3.4.7 –C .................................................................................................................................................................. 53
      3.4.8 –c .................................................................................................................................................................. 54
      3.4.9 –copyargs ...................................................................................................................................................... 54
      3.4.10 –Dname=[def] ............................................................................................................................................. 54
      3.4.11 –dalign .......................................................................................................................................................... 55
      3.4.12 –dbl_align_all=[yes|no] ........................................................................................................................ 56
      3.4.13 –depend=[yes|no] ....................................................................................................................................... 56
      3.4.14 –dn ................................................................................................................................................................. 57
      3.4.15 –dryrun ........................................................................................................................................................ 57
      3.4.16 –d{y|n} .......................................................................................................................................................... 57
      3.4.17 –e .................................................................................................................................................................. 57
      3.4.18 –erroff=[%all|%none|taglist] .................................................................................................................... 58
      3.4.19 –errtags=[yes|no] .......................................................................................................................................... 58
      3.4.20 –errwarn=[%all|%none|taglist] .................................................................................................................. 58
3.4.21 -ext_names=c ................................................................. 59
3.4.22 -F .................................................................................... 59
3.4.23 -f .................................................................................... 59
3.4.24 -f77[=list] ....................................................................... 60
3.4.25 -fast ................................................................................. 61
3.4.26 -fixed .............................................................................. 63
3.4.27 -flags ............................................................................. 63
3.4.28 -fma=[none|fused] ............................................................ 63
3.4.29 -fnonstd ......................................................................... 64
3.4.30 -fns=[{yes|no}] ................................................................. 64
3.4.31 -fpoer=[{yes|no}] .............................................................. 65
3.4.32 -fpp .................................................................................. 65
3.4.33 -fprecision=[single|double|extended] .............................. 66
3.4.34 -free ................................................................................ 66
3.4.35 -fround=[nearest|tozero|negative|positive] ....................... 66
3.4.36 -fsimple=[1|2|0] ................................................................. 67
3.4.37 -fstore .......................................................................... 68
3.4.38 -ftrap=t ......................................................................... 68
3.4.39 -G .................................................................................... 69
3.4.40 -g .................................................................................... 69
3.4.41 -hname .......................................................................... 70
3.4.42 -help ............................................................................... 70
3.4.43 -Ipath ............................................................................ 70
3.4.44 -i8 ................................................................................. 71
3.4.45 -inline=[auto][!,][no%][!,...[no%][!n] ............................. 71
3.4.46 -iorounding=[compatible|processor-defined] .................. 72
3.4.47 -Kpic .............................................................................. 72
3.4.48 -KPIC ............................................................................ 72
3.4.49 -Lpath .......................................................................... 72
3.4.50 -Lx ................................................................................ 73
3.4.51 -libmil ........................................................................... 73
3.4.52 -loopinfo ........................................................................ 74
3.4.53 -Mpath .......................................................................... 74
3.4.54 -m32 | -m64 .................................................................... 75
3.4.55 -moddir=path .................................................................. 75
3.4.56 -mt ............................................................................... 76
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.57 –native</td>
<td>76</td>
</tr>
<tr>
<td>3.4.58 –noautopar</td>
<td>76</td>
</tr>
<tr>
<td>3.4.59 –nolib</td>
<td>77</td>
</tr>
<tr>
<td>3.4.60 –nolibmil</td>
<td>77</td>
</tr>
<tr>
<td>3.4.61 –nofstore</td>
<td>77</td>
</tr>
<tr>
<td>3.4.62 –onethread</td>
<td>80</td>
</tr>
<tr>
<td>3.4.63 –onetrip</td>
<td>80</td>
</tr>
<tr>
<td>3.4.64 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.65 –pad</td>
<td>81</td>
</tr>
<tr>
<td>3.4.66 –openmp</td>
<td>81</td>
</tr>
<tr>
<td>3.4.67 –openmp</td>
<td>81</td>
</tr>
<tr>
<td>3.4.68 –openmp</td>
<td>81</td>
</tr>
<tr>
<td>3.4.69 –openmp</td>
<td>81</td>
</tr>
<tr>
<td>3.4.70 –openmp</td>
<td>81</td>
</tr>
<tr>
<td>3.4.71 –openmp</td>
<td>81</td>
</tr>
<tr>
<td>3.4.72 –o name</td>
<td>80</td>
</tr>
<tr>
<td>3.4.73 –onemodule</td>
<td>80</td>
</tr>
<tr>
<td>3.4.74 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.75 –onethread</td>
<td>80</td>
</tr>
<tr>
<td>3.4.76 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.77 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.78 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.79 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.80 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.81 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.82 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.83 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.84 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.85 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.86 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.87 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.88 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.89 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.90 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.91 –openmp</td>
<td>80</td>
</tr>
<tr>
<td>3.4.92 –openmp</td>
<td>80</td>
</tr>
</tbody>
</table>
3.4.93 \( \text{--temp=dir} \) .......................................................... \begin{array}{l} 87 \end{array} \\
3.4.94 \( \text{--time} \) ........................................................................ \begin{array}{l} 88 \end{array} \\
3.4.95 \( \text{--U} \) ........................................................................ \begin{array}{l} 88 \end{array} \\
3.4.96 \( \text{--Uname} \) .................................................................. \begin{array}{l} 88 \end{array} \\
3.4.97 \( \text{--u} \) .......................................................................... \begin{array}{l} 88 \end{array} \\
3.4.98 \( \text{--unroll=n} \) ........................................................... \begin{array}{l} 89 \end{array} \\
3.4.99 \( \text{--use=list} \) ................................................................ \begin{array}{l} 89 \end{array} \\
3.4.100 \( \text{--V} \) ...................................................................... \begin{array}{l} 89 \end{array} \\
3.4.101 \( \text{--v} \) ....................................................................... \begin{array}{l} 89 \end{array} \\
3.4.102 \( \text{--vax=keywords} \) ...................................................... \begin{array}{l} 89 \end{array} \\
3.4.103 \( \text{--vpara} \) .................................................................. \begin{array}{l} 90 \end{array} \\
3.4.104 \( \text{--w[n]} \) .................................................................. \begin{array}{l} 91 \end{array} \\
3.4.105 \( \text{--Xlist[x]} \) .............................................................. \begin{array}{l} 91 \end{array} \\
3.4.106 \( \text{--xa} \) ................................................................... \begin{array}{l} 92 \end{array} \\
3.4.107 \( \text{--xaddr32=[yes|no]} \) ............................................. \begin{array}{l} 92 \end{array} \\
3.4.108 \( \text{--xalias=[keywords]} \) ............................................ \begin{array}{l} 93 \end{array} \\
3.4.109 \( \text{--xannotate=[yes|no]} \) ............................................ \begin{array}{l} 95 \end{array} \\
3.4.110 \( \text{--xarch=isa} \) .......................................................... \begin{array}{l} 95 \end{array} \\
3.4.111 \( \text{--xassume_control=[keywords]} \) ......................... \begin{array}{l} 99 \end{array} \\
3.4.112 \( \text{--xautopar} \) ............................................................ \begin{array}{l} 100 \end{array} \\
3.4.113 \( \text{--xbinopt=prepare|off} \) ......................................... \begin{array}{l} 100 \end{array} \\
3.4.114 \( \text{--xcache=c} \) ......................................................... \begin{array}{l} 101 \end{array} \\
3.4.115 \( \text{--xcheck=keyword} \) ............................................. \begin{array}{l} 102 \end{array} \\
3.4.116 \( \text{--xchip=c} \) ............................................................ \begin{array}{l} 103 \end{array} \\
3.4.117 \( \text{--xcode=keyword} \) ................................................. \begin{array}{l} 104 \end{array} \\
3.4.118 \( \text{--xcommonchk=[yes|no]} \) .................................. \begin{array}{l} 105 \end{array} \\
3.4.119 \( \text{--xcrossfile=[1|0]} \) ............................................. \begin{array}{l} 106 \end{array} \\
3.4.120 \( \text{--xdebugformat=dwarf|stabs} \) ......................... \begin{array}{l} 106 \end{array} \\
3.4.121 \( \text{--xdepend} \) .......................................................... \begin{array}{l} 107 \end{array} \\
3.4.122 \( \text{--xF} \) ................................................................. \begin{array}{l} 107 \end{array} \\
3.4.123 \( \text{--xfilebyteorder=options} \) .................................. \begin{array}{l} 108 \end{array} \\
3.4.124 \( \text{--xhasc=[yes|no]} \) ................................................ \begin{array}{l} 110 \end{array} \\
3.4.125 \( \text{--xhelp=[readme|flags]} \) ................................... \begin{array}{l} 110 \end{array} \\
3.4.126 \( \text{--xhwprof=[enable|disable]} \) ......................... \begin{array}{l} 111 \end{array} \\
3.4.127 \( \text{--xia=[widestneed|strict]} \) ............................... \begin{array}{l} 111 \end{array} \\
3.4.128 \( \text{--xinline=list} \) .................................................. \begin{array}{l} 112 \end{array}
3.4.129 -xinstrument=[%no]datarace ................................................................. 112
3.4.130 -xinterval=[widestneed|strict|no] ............................................................ 112
3.4.131 -xipo=[0|1|2] ...................................................................................... 113
3.4.132 -xipo_archive=[none|readonly|writeback] ........................................ 115
3.4.133 -xjobs=N ............................................................................................. 116
3.4.134 -xknown_lib=library_list .................................................................. 117
3.4.135 -xlang=f77 .......................................................................................... 117
3.4.136 -xlibmil ............................................................................................... 118
3.4.137 -xlibmopt ........................................................................................... 118
3.4.138 -xlic_lib=sunperf ............................................................................... 118
3.4.139 -xlicinfo ............................................................................................. 118
3.4.140 -xlinkopt=[1|2|0] ............................................................................... 119
3.4.141 -xloopinfo ........................................................................................... 120
3.4.142 -xmaxopt=[n] ...................................................................................... 120
3.4.143 -xmemalign=<a><b> ........................................................................... 120
3.4.144 -xmodel=[small|kernel|medium] .......................................................... 122
3.4.145 -xnolb .................................................................................................. 122
3.4.146 -xnolibmil ............................................................................................ 122
3.4.147 -xnolbmopt ......................................................................................... 122
3.4.148 -xOn ..................................................................................................... 123
3.4.149 -xopenmp=[parallel|noopt|none] ......................................................... 123
3.4.150 -xpad ................................................................................................... 124
3.4.151 -xpagine=size ...................................................................................... 124
3.4.152 -xpagine_heap=size ........................................................................... 125
3.4.153 -xpagine_stack=size .......................................................................... 125
3.4.154 -xspec=[yes|no] .................................................................................. 125
3.4.155 -xpg ..................................................................................................... 126
3.4.156 -xpp=[fpp|cpp] .................................................................................... 126
3.4.157 -xprefetch=[a],[a] ............................................................................. 126
3.4.158 -xprefetch_auto_type=indirect_array_access .................................... 128
3.4.159 -xprefetch_level=[1|2|3] ................................................................. 129
3.4.160 -xprofile=[collect[:name]|use[:name]|tcov] .................................. 129
3.4.161 -xprofile_ircache=[path] ................................................................. 131
3.4.162 -xprofile_pathmap=[collect_prefix]:use_prefix ............................... 132
3.4.163 -xrecursive ........................................................................................ 132
3.4.164 -xreduction ....................................................................................... 133
3.4.165 –xregs=r ............................................................ 133
3.4.166 –xs ........................................................................ 134
3.4.167 –xsafe=mem ......................................................... 134
3.4.168 –xsb ..................................................................... 134
3.4.169 –xsbfast .............................................................. 134
3.4.170 –xspace .................................................................. 134
3.4.171 –xtarget= ................................................................ 135
3.4.172 –xtime .................................................................... 137
3.4.173 –xtypemap=spec .................................................. 137
3.4.174 –xunroll=n ............................................................ 138
3.4.175 –xvector=[ [no%]lib,[no%]simd,%none] ....................... 138
3.4.176 –ztext .................................................................... 139

4 Sun Studio Fortran Features and Differences .................................. 141

4.1 Source Language Features .............................................. 141
4.1.1 Continuation Line Limits ............................................ 141
4.1.2 Fixed-Form Source Lines ........................................... 141
4.1.3 Tab Form ................................................................. 141
4.1.4 Source Form Assumed ................................................ 142
4.1.5 Limits and Defaults .................................................... 143

4.2 Data Types ................................................................... 143
4.2.1 Boolean Type ............................................................ 143
4.2.2 Abbreviated Size Notation for Numeric Data Types .......... 146
4.2.3 Size and Alignment of Data Types ............................... 147

4.3 Cray Pointers ............................................................... 149
4.3.1 Syntax ...................................................................... 149
4.3.2 Purpose of Cray Pointers ........................................... 150
4.3.3 Declaring Cray Pointers and Fortran 95 Pointers ............ 150
4.3.4 Features of Cray Pointers .......................................... 150
4.3.5 Restrictions on Cray Pointers ..................................... 151
4.3.6 Restrictions on Cray Pointees ..................................... 151
4.3.7 Usage of Cray Pointers .............................................. 151

4.4 STRUCTURE and UNION (VAX Fortran) ..................... 152

4.5 Unsigned Integers ......................................................... 153
4.5.1 Arithmetic Expressions ............................................. 154
4.5.2 Relational Expressions ................................................................. 154
4.5.3 Control Constructs ................................................................. 154
4.5.4 Input/Output Constructs .......................................................... 154
4.5.5 Intrinsic Functions ................................................................. 155
4.6 Fortran 2003 Features ................................................................. 155
   4.6.1 Interoperability with C Functions ........................................ 155
   4.6.2 IEEE Floating-Point Exception Handling .......................... 156
   4.6.3 Command-Line Argument Intrinsics ................................. 156
   4.6.4 PROTECTED Attribute ......................................................... 156
   4.6.5 Fortran 2003 Asynchronous I/O ........................................ 156
   4.6.6 Extended ALLOCATABLE Attribute ............................... 157
   4.6.7 VALUE Attribute ................................................................. 157
   4.6.8 Fortran 2003 Stream I/O ..................................................... 158
   4.6.9 Fortran 2003 Formatted I/O Features ............................... 158
   4.6.10 Fortran 2003 IMPORT Statement ..................................... 159
   4.6.11 Fortran 2003 FLUSH I/O Statement ................................. 159
4.7 Additional I/O Extensions .......................................................... 159
   4.7.1 I/O Error Handling Routines ............................................ 159
   4.7.2 Variable Format Expressions ............................................ 160
   4.7.3 NAMELIST Input Format .................................................. 160
   4.7.4 Binary Unformatted I/O .................................................. 160
   4.7.5 Miscellaneous I/O Extensions .......................................... 161
4.8 Directives .................................................................................. 161
   4.8.1 Form of Special f95 Directive Lines ................................. 161
   4.8.2 FIXED and FREE Directives ............................................ 162
   4.8.3 Parallelization Directives .................................................. 163
4.9 Module Files ................................................................................ 163
   4.9.1 Searching for Modules ..................................................... 164
   4.9.2 The -use=list Option Flag ............................................. 164
   4.9.3 The fdumpmod Command ............................................ 165
4.10 Intrinsics .................................................................................. 165
4.11 Forward Compatibility ......................................................... 166
4.12 Mixing Languages ................................................................. 166
Preface

The *Fortran User’s Guide* describes the environment and command-line options for the Sun Studio Fortran compiler, *f95*. 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 compiler effectively. Familiarity with the Solaris operating environment or UNIX® in general is also assumed.

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

**Typographic Conventions**

The following table describes the typographic conventions that are used in this book.

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories, and onscreen computer output</td>
<td>Edit your .login file. Use <code>ls -a</code> to list all files. `machine_name% you have mail.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, contrasted with onscreen computer output</td>
<td><code>machine_name% su</code> Password:</td>
</tr>
<tr>
<td>aabbcc123</td>
<td>Placeholder: replace with a real name or value</td>
<td>The command to remove a file is <code>rm filename</code>.</td>
</tr>
</tbody>
</table>


TABLE P–1  Typographic Conventions (Continued)

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new terms, and terms to be emphasized</td>
<td>Read Chapter 6 in the User’s Guide. A cache is a copy that is stored locally. Do not save the file. Note: Some emphasized items appear bold online.</td>
</tr>
</tbody>
</table>

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

∇∇36.001

- The FORTRAN 77 standard used an older convention, spelling the name “FORTRAN” capitalized. The current convention is to use lower case: “Fortran”

- References to online man pages appear with the topic name and section number. For example, a reference to the library routine GETENV will appear as getenv(3F), implying that the man command to access this man page would be: man -s 3F getenv

Shell Prompts in Command Examples

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

TABLE P–2  Shell Prompts

<table>
<thead>
<tr>
<th>Shell</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C shell</td>
<td>machine_name%</td>
</tr>
<tr>
<td>C shell for superuser</td>
<td>machine_name#</td>
</tr>
<tr>
<td>Bourne shell and Korn shell</td>
<td>$</td>
</tr>
<tr>
<td>Bourne shell and Korn shell for superuser</td>
<td>#</td>
</tr>
</tbody>
</table>

Supported Platforms

This Sun™ Studio release supports systems that use the SPARC® and x86 families of processor architectures: UltraSPARC®, SPARC64, AMD64, Pentium, and Xeon EM64T. The supported systems for the version of the Solaris Operating System you are running are available in the hardware compatibility lists at http://www.sun.com/bigadmin/hcl. These documents cite any implementation differences between the platform types.

In this document, these x86 related terms mean the following:
“x86” refers to the larger family of 64-bit and 32-bit x86 compatible products.
“x64” points out specific 64-bit information about AMD64 or EM64T systems.
“32-bit x86” points out specific 32-bit information about x86 based systems.

For supported systems, see the hardware compatibility lists.

Accessing Sun Studio Documentation

You can access the documentation at the following locations:

- Online help for all components of the IDE is available through the Help menu, as well as through Help buttons on many windows and dialog boxes, in the IDE.
- Online help for the Performance Analyzer is available through the Help menu, as well as through Help buttons on many windows and dialog boxes, in the Performance Analyzer.

The docs.sun.com web site ([http://docs.sun.com](http://docs.sun.com)) enables you to read, print, and buy Sun Microsystems manuals through the Internet.

**Note** – Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation in Accessible Formats

The documentation is provided in accessible formats that are readable by assistive technologies for users with disabilities. You can find accessible versions of documentation as described in the following table.

<table>
<thead>
<tr>
<th>Type of Documentation</th>
<th>Format and Location of Accessible Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuals</td>
<td>HTML at <a href="http://docs.sun.com">http://docs.sun.com</a></td>
</tr>
</tbody>
</table>
Resources for Developers

Visit http://developers.sun.com/sunstudio to find these frequently updated resources:

- Articles on programming techniques and best practices
- Documentation of the software, as well as corrections to the documentation that is installed with your software
- Tutorials that take you step-by-step through development tasks using Sun Studio tools
- Information on support levels
- User forums
- Downloadable code samples
- New technology previews

The Sun Studio portal is one of a number of additional resources for developers at the Sun Developer Network web site, http://developers.sun.com.

Contacting Technical Support

If you have technical questions about this product that are not answered in this document, go to http://www.sun.com/service/contacting

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Submit your comments to Sun at this URL: http://www.sun.com/hwdocs/feedback.

Please include the part number of the document in the subject line of your email. For example, the part number for this document is 820-7600-10.
The Sun™ Studio Fortran compiler, f95, described here and in the companion Fortran Programming Guide, is available under the Solaris™ operating environment on SPARC®, UltraSPARC®, and x64/x86 platforms, and Linux environments on x86/x64 platforms. The compiler conforms to published Fortran language standards, and provides many extended features, including multiprocessor parallelization, sophisticated optimized code compilation, and mixed C/Fortran language support.

The f95 compiler also provides a Fortran 77 compatibility mode that accepts most legacy Fortran 77 source codes. A separate Fortran 77 compiler is no longer included. See Chapter 5 for information on FORTRAN 77 compatibility and migration issues.

1.1 Standards Conformance

- f95 conforms to part one of the ISO/IEC 1539-1:1997 Fortran standards document.
- f95 provides support for the optimization-exploiting features of the SPARC and x86 families of processor architectures: UltraSPARC, SPARC64, AMD64, Pentium Pro, and Xeon Intel® 64, on Solaris and Linux (x86) platforms.
- Sun Studio compilers conform to the OpenMP 3.0 shared memory parallelization API specifications. See the OpenMP API User’s Guide for details.
- 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.
1.2 Features of the Fortran Compiler

The Sun Studio Fortran compiler, f95, provides the following features and extensions:

- Global program checking across routines for consistency of arguments, commons, parameters, and the like. (Only available on Solaris platforms).
- Optimized automatic and explicit loop parallelization for multiprocessor systems.
- VAX/VMS Fortran extensions, including:
  - Structures, records, unions, maps
  - Recursion
- OpenMP 3.0 parallelization directives.
- 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 permit routines written in C or C++ to be combined with Fortran programs.
- Support for 64-bit enabled Solaris and Linux environments.
- Call-by-value using %VAL.
- Compatibility between Fortran 77 and Fortran 95/Fortran 2003 programs and object binaries.
- Interval Arithmetic programming.
- Some Fortran 2003 features, including Stream I/O.

See Appendix B for details on new and extended features added to the compiler with each software release.

1.3 Other Fortran Utilities

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

- Sun Studio Performance Analyzer— In depth performance analysis tool for single threaded and multi-threaded applications. See analyzer(1).
- 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).
- fdumpmod— A utility to display the names of modules contained in a file or archive. See fdumpmod(1).
- fpp— A Fortran source code preprocessor. See fpp(1).
- **fsplit** — This utility splits one Fortran file of several routines into several files, each with one routine per file. See *fsplit*(1)

### 1.4 Debugging Utilities

The following debugging utilities are available:

- **-Xlist** — A compiler option to check across routines for consistency of arguments, COMMON blocks, and so on. (Solaris platforms only)

- **Sun Studio dbx** — Provides a robust and feature-rich runtime and static debugger, and includes a performance data collector.

### 1.5 Sun Performance Library

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 LAPACK, BLAS1, BLAS2, BLAS3, FTFPACK, VFFTBACK, and LINPACK generally available through Netlib ([www.netlib.org](http://www.netlib.org)).

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 accurate and can be used in a multiprocessing environment.

See the `performance_library` README file, and the *Sun Performance Library User’s Guide* for details. (Man pages for the performance library routines are in section 3P.)

### 1.6 Interval Arithmetic

The *f95* compiler provides the compiler flags `-xia` and `-xinterval` to enable new language extensions and generate the appropriate code to implement interval arithmetic computations. See the *Fortran 95 Interval Arithmetic Programming Reference* for details. (Interval arithmetic features are only supported on SPARC/UltraSPARC platforms.)

### 1.7 Man Pages

Online manual (`man`) pages provide immediate documentation about a command, function, subroutine, or collection of such things. The user's `MANPATH` environment variable should be set to the path to the installed Sun Studio `man` directory to access the Sun Studio `man` pages.

You can display a man page by running the command:
Throughout the Fortran documentation, man page references appear with the topic name and man section number: \texttt{f95(1)} is accessed with \texttt{man f95}. Other sections, denoted by \texttt{ieee\_flags(3M)} for example, are accessed using the \texttt{-s} option on the \texttt{man} command:

\begin{verbatim}
demo\% man -s 3M ieee\_flags
\end{verbatim}

The Fortran library routines are documented in the man page section 3F.

The following lists man pages of interest to Fortran users:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{f95(1)}</td>
<td>The Fortran 95 command-line options</td>
</tr>
<tr>
<td>\texttt{analyzer(1)}</td>
<td>Sun Studio Performance Analyzer</td>
</tr>
<tr>
<td>\texttt{asa(1)}</td>
<td>Fortran carriage-control print output post-processor</td>
</tr>
<tr>
<td>\texttt{dbx(1)}</td>
<td>Command-line interactive debugger</td>
</tr>
<tr>
<td>\texttt{fpp(1)}</td>
<td>Fortran source code pre-processor</td>
</tr>
<tr>
<td>\texttt{cpp(1)}</td>
<td>C source code pre-processor</td>
</tr>
<tr>
<td>\texttt{fdumpmod(1)}</td>
<td>Display contents of a MODULE (.mod) file.</td>
</tr>
<tr>
<td>\texttt{fsplit(1)}</td>
<td>Pre-processor splits Fortran source routines into single files</td>
</tr>
<tr>
<td>\texttt{ieee_flags(3M)}</td>
<td>Examine, set, or clear floating-point exception bits</td>
</tr>
<tr>
<td>\texttt{ieee_handler(3M)}</td>
<td>Handle floating-point exceptions</td>
</tr>
<tr>
<td>\texttt{matherr(3M)}</td>
<td>Math library error handling routine</td>
</tr>
<tr>
<td>\texttt{ld(1)}</td>
<td>Link editor for object files</td>
</tr>
</tbody>
</table>

### 1.8 README Files

The \texttt{README} pages on the Sun Developer Network (SDN) portal (http://developers.sun.com/sunstudio/) describe new features, software incompatibilities, bugs, and information that was discovered after the manuals were printed. These \texttt{README} pages are part of the documentation on the portal for this release, and are also linked from the HTML documentation index that is part of the installed software at file:/opt/SUNWspro/docs.
TABLE 1–1  README Pages of Interest

<table>
<thead>
<tr>
<th>README Page</th>
<th>Describes...</th>
</tr>
</thead>
<tbody>
<tr>
<td>fortran_95</td>
<td>new and changed features, known limitations, documentation errata for this release of the Fortran compiler, f95.</td>
</tr>
<tr>
<td>fpp_readme</td>
<td>overview of fpp features and capabilities</td>
</tr>
<tr>
<td>interval_arithmetic</td>
<td>overview of the interval arithmetic features in f95</td>
</tr>
<tr>
<td>math_libraries</td>
<td>optimized and specialized math libraries available.</td>
</tr>
<tr>
<td>profiling_tools</td>
<td>using the performance profiling tools, prof, gprof, and tcov.</td>
</tr>
<tr>
<td>runtime_libraries</td>
<td>libraries and executables that can be redistributed under the terms of the End User License.</td>
</tr>
<tr>
<td>performance_library</td>
<td>overview of the Sun Performance Library</td>
</tr>
<tr>
<td>openmp</td>
<td>new and changed features in the OpenMP parallelization API</td>
</tr>
</tbody>
</table>

The URL to the README page is displayed by the -xhelp=readme command-line option. For example, the command:

```bash
% f95 -xhelp=readme
```

displays the URL for viewing the fortran_95 README for this release on the SDN portal.

## 1.9 Command-Line Help

You can view very brief descriptions of the f95 command line options by invoking the compiler’s -help option as shown below:

```bash
%f95 -help=flags
```

Items within [ ] are optional. Items within < > are variable parameters.  
Bar | indicates choice of literal values.  
-someoption={yes|no} implies -someoption is equivalent to -someoption=yes

-`a` Collect data for tcov basic block profiling  
-`aligncommon=[<a>]` Align common block elements to the specified boundary requirement; `<a>`={1|2|4|8|16}  
-`ansi` Report non-ANSI extensions.  
-`autopar` Enable automatic loop parallelization  
-`bdynamic` Allow dynamic linking  
-`bstatic` Require static linking  
-`C` Enable runtime subscript range checking  
-`c` Compile only; produce .o files but suppress
1.9 Command-Line Help

linking

...etc.
Using Sun Studio Fortran

This chapter describes how to use the Fortran compiler.

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 compiler can also be used to:

- Generate a parallelized executable file for multiple processors (-openmp).
- 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 files into an executable file.
- Compile an executable file with runtime debugging enabled (-g).
- Compile with runtime statement or procedure level profiling (-pg).
- Check source code for ANSI standards conformance (-ansi).

2.1 A Quick Start

This section provides a quick overview of how to use the Fortran compiler to compile and run Fortran programs. A full reference to command-line options appears in the next chapter.
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`, `.F90`, or `.F95` 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:

```fortran
demo% cat greetings.f
PROGRAM GREETINGS
  PRINT *, 'Real programmers write Fortran!'
END
```

demo% f95 greetings.f
demo% a.out
  Real programmers write Fortran!
demo%

In this example, `f95` compiles source file `greetings.f` and links the executable program onto the file, `a.out`, by default. To launch the 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:

```fortran
demo% f95 -o greetings greetings.f
demo% greetings
  Real programmers write Fortran!
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 at a shell prompt.

The next sections of this chapter discuss the conventions used by the `f95` commands, compiler source line directives, and other issues concerning the use of these compiler. The next chapter describes the command-line syntax and all the options in detail.
2.2 Invoking the Compiler

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

```sh
f95 [options] files...
```

Here `files...` is one or more Fortran source file names ending in `.f`, `.F`, `.f90`, `.f95`, `.F90`, `.F95`, or `.for`; `options` is one or more of the compiler option flags. (Files with names ending in a `.f90` or `.f95` extension are “free-format” Fortran 95 source files recognized only by the `f95` compiler.)

In the example below, `f95` is used to compile two source files to produce an executable file named `growth` with runtime debugging enabled:

```sh
demo% f95 -g -o growth growth.f fft.f95
```

Note – You can invoke the Fortran compiler with either the `f95` or `f90` command.

New: The compiler will also accept source files with the extension `.f03` or `.F03`. These are treated as equivalent to `.f95` and `.F95` and could be used as a way to indicate that a source file contains Fortran 2003 extensions.

“2.2.2 Command-Line File Name Conventions” on page 25, describes the various source file extensions accepted by the compiler.

2.2.1 Compile-Link Sequence

In the previous example, the compiler automatically generates the loader object files, `growth.o` and `fft.o`, and then invokes the system linker to create the executable program 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 file is written.

2.2.2 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.
2.2 Invoking the Compiler

### TABLE 2–1 Filename Suffixes Recognized by the Fortran Compiler

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Language</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>.f</td>
<td>Fortran 77 or Fortran 95</td>
<td>Compile Fortran source files, put object files in current directory; default name of object file is that of the source but with .o suffix.</td>
</tr>
<tr>
<td>.f95.f90</td>
<td>Fortran 95</td>
<td>Same action as .f</td>
</tr>
<tr>
<td>.f03</td>
<td>Fortran 2003</td>
<td>Same action as .f</td>
</tr>
<tr>
<td>.for</td>
<td>Fortran 77 or Fortran 95</td>
<td>Same action as .f</td>
</tr>
<tr>
<td>.F</td>
<td>Fortran 77 or Fortran 95</td>
<td>Apply the Fortran (or C) preprocessor to the Fortran 77 source file before compilation.</td>
</tr>
<tr>
<td>.F95.F90</td>
<td>Fortran 95</td>
<td>Apply the Fortran (or C) preprocessor to the Fortran 95 free-format source file before Fortran compiles it.</td>
</tr>
<tr>
<td>.F03</td>
<td>Fortran 2003</td>
<td>Same as .F95</td>
</tr>
<tr>
<td>.s</td>
<td>Assembler</td>
<td>Assemble source files with the assembler.</td>
</tr>
<tr>
<td>.S</td>
<td>Assembler</td>
<td>Apply the C preprocessor to the assembler source file before assembling it.</td>
</tr>
<tr>
<td>.il</td>
<td>Inline expansion</td>
<td>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 <code>inline(1)</code> man page.)</td>
</tr>
<tr>
<td>.o</td>
<td>Object files</td>
<td>Pass object files through to the linker.</td>
</tr>
<tr>
<td>.a,.o,.so</td>
<td>Libraries</td>
<td>Pass names of libraries to the linker. .a files are static libraries, .so and .so.n files are dynamic libraries.</td>
</tr>
</tbody>
</table>

Fortran 95 free-format is described in "4.1 Source Language Features" on page 141.

### 2.2.3 Source Files

The Fortran compiler 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). Applications may be configured with one source code procedure per file, or by gathering procedures that work together into single files. The *Fortran Programming Guide* describes the advantages and disadvantages of these configurations.
2.2.4 Source File Preprocessors

**f95** supports 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 compiler 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 Fortran-specific source preprocessor. See the **fpp**(1) man page and the **fpp** README for details. It is invoked by default on files with a **.F**, **.F90**, **.F95**, or **.F03** extension.

The source code for **fpp** is available from the Netlib website at

http://www.netlib.org/fortran/

See **cpp**(1) for information on the standard Unix C language preprocessor. Use of **fpp** over **cpp** is recommended on Fortran source files.

2.2.5 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% f95 -c file1.f (Make new object file)
demo% f95 -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).

2.2.6 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 some options requires linking with the same options. Also, a number of options require that all source files be compiled with that option, including the link step.

The option descriptions in Chapter 3 identify such options.

Example: Compiling **sbr.f** with **-fast**, compiling a C routine, and then linking in a separate step:
2.2 Invoking the Compiler

```
demo% f95 -c -fast sbr.f

demo% cc -c -fast simm.c

demo% f95 -fast sbr.o simm.o
```

link step; passes -fast to the linker

### 2.2.7 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% f95 -bit move.f
```

<- `bit` is not a recognized `f95` option

```
f95: Warning: Option -bit passed to ld, if ld is invoked, ignored otherwise
```

```
demo% f95 fast move.f
```

<- The user meant to type `-fast`

```
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 `f95` 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 `f95` 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!

### 2.2.8 Modules

`f95` automatically creates module information files for each **MODULE** declaration encountered in the source files, and searches for modules referenced by a **USE** statement. For each module encountered (**MODULE module_name**), the compiler generates a corresponding file, `module_name.mod`, in the current directory. For example, `f95` generates the module information file `list.mod` for the **MODULE list** unit found on file `mysrc.f95`.

See the `-Mpath` and `-moddir dirlist` option flags for information on how to set the defaults paths for writing and searching for module information files.
See also the -use compiler option for implicitly invoking MODULE declarations in all compilation units.

Use the fdumpmod(1) command to display information about the contents of a .mod module information file.

For detailed information, see “4.9 Module Files” on page 163.

2.3 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 sometimes called pragmas. The compiler recognize a set of general directives and parallelization directives. f95 also processes OpenMP shared memory multiprocessing directives.

Directives unique to f95 are described in “4.8 Directives” on page 161. A complete summary of all the directives recognized by f95 appears in Table C–1.

Note – Directives are not part of the Fortran standard.

2.3.1 General Directives

The various forms of a general Fortran directive are:

\[
\text{C$PRAGMA \text{keyword} ( a [, a ] \ldots ) [ , \text{keyword} ( a [, a ] \ldots ) ] , \ldots}
\]

\[
\text{C$PRAGMA SUN \text{keyword} ( a [, a ] \ldots ) [ , \text{keyword} ( a [, a ] \ldots ) ] , \ldots}
\]

\[
\text{C$PRAGMA SPARC \text{keyword} ( a [, a ] \ldots ) [ , \text{keyword} ( a [, a ] \ldots ) ] , \ldots}
\]

The variable keyword identifies the specific directive. Additional arguments or suboptions may also be allowed. (Some directives require the additional keyword SUN or SPARC, as shown above.)

A general directive has the following syntax:

- In column one, any of the comment-indicator characters c, C, !, or *.
- For f95 free-format, ! is the only comment-indicator recognized ( !$PRAGMA ). The examples in this chapter assume fixed-format.
- The next seven characters are $PRAGMA, no blanks, in either uppercase or lowercase.
- Directives using the ! comment-indicator character may appear in any position on the line for free-format source programs.

Observe the following restrictions:
2.3 Directives

- 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. fpp will recognize legitimate f95 directives and allow limited substitution outside directive keywords. However, be careful with directives requiring the keyword SUN. cpp will replace lower-case sun with a predefined value. Also, if you define a cpp macro SUN, it might interfere with the SUN directive keyword. A general rule would 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 Fortran compiler recognize the following general directives:

<table>
<thead>
<tr>
<th>C Directive</th>
<th>C$PRAGMA C(list)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Declares a list of names of external functions as C language routines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IGNORE_TKR Directive</th>
<th>C$PRAGMA IGNORE_TKR(name [, name] ...)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The compiler ignores the type, kind, and rank of the specified dummy argument names appearing in a generic procedure interface when resolving a specific call.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNROLL Directive</th>
<th>C$PRAGMA SUN UNROLL=m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Advises the compiler that the following loop can be unrolled to a length n.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WEAK Directive</th>
<th>C$PRAGMA WEAK(name=name2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Declares name to be a weak symbol, or an alias for name2.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OPT Directive</th>
<th>C$PRAGMA SUN OPT=n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Set optimization level for a subprogram to n.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PIPELOOP Directive</th>
<th>C$PRAGMA SUN PIPELOOP=n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assert dependency in the following loop exists between iterations n apart.</td>
</tr>
</tbody>
</table>
TABLE 2–2  Summary of General Fortran Directives  (Continued)

<table>
<thead>
<tr>
<th>PREFETCH Directives</th>
<th>C$PRAGMA SUN_PREFETCH_READ_ONCE(name)</th>
<th>SUN_PREFETCH_WRITE_ONCE(name)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C$PRAGMA SUN_PREFETCH_READ_MANY(name)</td>
<td>C$PRAGMA SUN_PREFETCH_WRITE_MANY(name)</td>
</tr>
<tr>
<td></td>
<td>Request compiler generate prefetch instructions for references to name.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Requires -xprefetch option, which is enabled by default. Prefetch directives can be disabled by compiling with -xprefetch=no. Target architecture must also support prefetch instructions, and the compiler optimization level must be greater than -xO2.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASSUME Directives</th>
<th>C$PRAGMA [BEGIN] ASSUME (expression [, probability])</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C$PRAGMA END ASSUME</td>
</tr>
<tr>
<td></td>
<td>Make assertions about conditions at certain points in the program that the compiler can assume are true.</td>
</tr>
</tbody>
</table>

2.3.1.1 The C Directive

The C() directive specifies that its arguments are external functions. 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 C-Fortran Interface chapter in the 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)
```

2.3.1.2 The IGNORE_TKR Directive

This directive causes the compiler to ignore the type, kind, and rank of the specified dummy argument names appearing in a generic procedure interface when resolving a specific call.

For example, in the procedure interface below, the directive specifies that SRC can be any data type, but LEN can be either KIND=4 or KIND=8. The interface block defines two specific procedures for a generic procedure name. This example is shown in Fortran 95 free format.

```
INTERFACE BLCKX
SUBROUTINE BLCK_32(LEN, SRC)
  REAL SRC(1)
!$PRAGMA IGNORE_TKR SRC
  INTEGER (KIND=4) LEN
```

The subroutine call:

```fortran
INTEGER L
REAL S(100)
CALL BLCKX(L, S)
```

The call to `BLCKX` will call `BLCK_32` when compiled normally, and `BLCK_64` when compiled with `-xtypemap=integer:64`. The actual type of `S` does not determine which routine to call. This greatly simplifies writing generic interfaces for wrappers that call specific library routines based on argument type, kind, or rank.

Note that dummy arguments for assumed-shape arrays, Fortran pointers, or allocatable arrays cannot be specified on the directive. If no names are specified, the directive applies to all dummy arguments to the procedure, except dummy arguments that are assumed-shape arrays, Fortran pointers, or allocatable arrays.

### 2.3.1.3 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 the following loop `n` times during its optimization pass. (The compiler will unroll a loop only when its analysis regards such unrolling as appropriate.)

`n` is a positive integer. The choices are:

- If `n=1`, the optimizer may not unroll any loops.
- If `n>1`, the optimizer may 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
```
2.3.1.4 The **WEAK** Directive

The **WEAK** directive defines a symbol to have less precedence than an earlier definition of the same symbol. This pragma is used mainly in source files for building libraries. The linker does not produce an error message if it is unable to resolve a weak symbol.

\[ \text{C\$PRAGMA WEAK ( } \text{name1} \[=\text{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.

2.3.1.5 The **OPT** Directive

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:

\[ \text{C\$PRAGMA SUN OPT=2} \]

| SUBROUTINE smart (a, b, c, d, e) |
| ...etc |

When the above is compiled with an f95 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.

2.3.1.6 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 dependence between loop iterations. A value of zero indicates that the loop has no inter-iteration (loop-carried) 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. (Default if \( n \) is not specified is 0)

C We know that the value of \( K \) is such that there can be no
C cross-iteration dependencies (E.g. \( K \gg N \))
C

```fortran
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*.

### 2.3.1.7 The **PREFETCH** Directives

The `-xprefetch` option flag, “3.4.157 –xprefetch[\(=a,[a]\)]” on page 126, enables a set of `PREFETCH` directives that advise the compiler to generate prefetch instructions for the specified data element on processors that support prefetch.

```fortran
C$PRAGMA SUN_PREFETCH_READ_ONCE(name)
C$PRAGMA SUN_PREFETCH_READ_MANY(name)
C$PRAGMA SUN_PREFETCH_WRITE_ONCE(name)
C$PRAGMA SUN_PREFETCH_WRITE_MANY(name)
```

See also the *C User’s Guide*, or the *SPARC Architecture Manual, Version 9* for further information about prefetch instructions.

### 2.3.1.8 The **ASSUME** Directives

The `ASSUME` directive gives the compiler hints about conditions at certain points in the program. These assertions can help the compiler to guide its optimization strategies. The programmer can also use these directives to check the validity of the program during execution. There are two formats for `ASSUME`.

The syntax of the “point assertion” `ASSUME` is

```fortran
C$PRAGMA ASSUME (expression [, probability])
```

Alternatively, the “range assertion” `ASSUME` is:

```fortran
C$PRAGMA BEGIN ASSUME [expression [, probability)]
   block of statements
C$PRAGMA END ASSUME
```

Use the point assertion form to state a condition that the compiler can assume at that point in the program. Use the range assertion form to state a condition that holds over the enclosed range of statements. The `BEGIN` and `END` pairs in a range assertion must be properly nested.
The required expression is a boolean expression that can be evaluated at that point in the program that does not involve user-defined operators or function calls except for those listed below.

The optional probability value is a real number from 0.0 to 1.0, or an integer 0 or 1, giving the probability of the expression being true. A probability of 0.0 (or 0) means never true, and 1.0 (or 1) means always true. If not specified, the expression is considered to be true with a high probability, but not a certainty. An assertion with a probability other than exactly 0 or 1 is a non-certain assertion. Similarly, an assertion with a probability expressed exactly as 0 or 1 is a certain assertion.

For example, if the programmer knows that the length of a DO loop is always greater than 10,000, giving this hint to the compiler can enable it to produce better code. The following loop will generally run faster with the ASSUME pragma than without it.

```fortran
C$Pragma begin assume(__tripcount().ge.10000,1) !! a big loop
   do i = j, n
      a(i) = a(j) + 1
   end do
C$Pragma end assume
```

Two intrinsic functions are available for use specifically in the expression clause of the ASSUME directive. (Note that their names are prefixed by two underscores.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__branchexp()</td>
<td>Use in point assertions placed immediately before a branching statement with a boolean controlling expression. It yields the same result as the boolean expression controlling the branching statement.</td>
</tr>
<tr>
<td>__tripcount()</td>
<td>Yields the trip count of the loop immediately following or enclosed by the directive. When used in a point assertion, the statement following the directive must be the first line of a DO. When used in a range assertion, it applies to the outermost enclosed loop.</td>
</tr>
</tbody>
</table>

This list of special intrinsics might expand in future releases.

Use with the -xassume_control compiler option. (See “3.4.111 -xassume_control[=keywords]” on page 99) For example, when compiled with -xassume_control=check, the example above would produce a warning if the trip count ever became less than 10,000.

Compiling with -xassume_control=retrospective will generate a summary report at program termination of the truth or falsity of all assertions. See the f95 man page for details on -xassume_control.

Another example:
Compiling the above example with -xassume_control=check will issue a runtime warning should the loop not be taken because the trip count is zero or negative.

2.3.2 Parallelization Directives

Parallelization directives explicitly request the compiler to attempt to parallelize the DO loop or the region of code that follows the directive. The syntax differs from general directives. Parallelization directives are only recognized when compiling with -openmp. Details regarding Fortran parallelization can be found in the OpenMP API User’s Guide and the Fortran Programming Guide.

The Fortran compiler supports the OpenMP 3.0 shared memory parallelization model. Legacy Sun and Cray parallelization directives are now deprecated and should not be used.

2.3.2.1 OpenMP Parallelization Directives

The Fortran compiler recognizes the OpenMP Fortran shared memory multiprocessing API as the preferred parallel programming model. The API is specified by the OpenMP Architecture Review Board (http://www.openmp.org).

You must compile with the command-line option -xopenmp, to enable OpenMP directives. (see “3.4.149 -xopenmp=[parallel|noopt|none]” on page 123.)

For more information about the OpenMP directives accepted by f95, see the OpenMP API User’s Guide.

2.3.2.2 Legacy Sun/Cray Parallelization Directives

Note – Legacy Sun and Cray style parallelization directives are now deprecated. The OpenMP parallelization API is preferred. Information on how to migrate from legacy Sun/Cray directives to the OpenMP model appears in the OpenMP API User’s Guide.

2.4 Library Interfaces and system.inc

The Fortran compiler provides an include file, system.inc, that defines the interfaces for most non-intrinsic library routines. Declare this include file to insure that functions you call and their arguments are properly typed, especially when default data types are changed with -xtypemap.

For example, the following may produce an arithmetic exception because function getpid() is not explicitly typed:
integer(4) mypid
mypid = getpid()
print *, mypid

The `getpid()` routine returns an integer value but the compiler assumes it returns a real value if no explicit type is declared for the function. This value is further converted to integer, most likely producing a floating-point error.

To correct this you should explicitly type `getuid()` and functions like it that you call:

```fortran
integer(4) mypid, getpid
mypid = getpid()
print *, mypid
```

Problems like these can be diagnosed with the `-Xlist` (global program checking) option. The Fortran include file `system.inc` provides explicit interface definitions for these routines.

```fortran
include 'system.inc'
integer(4) mypid
mypid = getpid()
print *, mypid
```

Including `system.inc` in program units calling routines in the Fortran library will automatically define the interfaces for you, and help the compiler diagnose type mismatches. (See the Fortran Library Reference for more information.)

## 2.5 Compiler Usage Tips

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

### 2.5.1 Determining Hardware Platform

Some compiler flags allow the user to tune code generation to a specific set of hardware platform options. The compiler's `-dryrun` option can be used to determine the native processor:

```
<sparc>%f95 -dryrun -xtarget=native
### command line files and options (expanded):
### -dryrun -xarch=v8plusb -xcache=64/32/4:1024/64/4 -xchip=ultra3i
```

```
<x64>%f95 -dryrun -xtarget=native
### command line files and options (expanded):
### -dryrun -xarch=ss6e2a -xcache=64/64/2:1024/64/16 -xchip=opteron
```
2.5.2 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 the implicit compilation rules of `make`, `FFLAGS` is used automatically by the `make` program.

Example: Set `FFLAGS`: (C Shell)

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

Example: Use `FFLAGS` explicitly:

```
demo% f95 $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 compiler command line, then invoking `make` will result in a compilation equivalent to:

```
f95 -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.

Note – Default implicit rules assumed by `make` may not recognize files with extensions `.f95` and `.mod` (Module files). See the Fortran Programming Guide and the Fortran readme file for details.

2.5.3 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 `-O` option on the command line.

A processor running the compiler should have at least 64 megabytes of memory; 256 megabytes are recommended. Enough swap space should also be allocated. 200 megabytes is the minimum; 300 megabytes is 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)`.

### 2.5.3.1 Swap Space Limits

The SunOS™ operating system 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
```

### 2.5.3.2 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
```

### 2.5.3.3 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 additional memory that could degrade compile-time performance. 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:

```
demos ulimit -d 16000
```

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

Example: Limit virtual memory to 16 Mbytes:

```
demos 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:
```
$ ulimit -d 1600
```

In a $csh$ shell:
```
% limit datasize 16M
```

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

In 64-bit Solaris 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 $csh$ use:
```
% limit datasize unlimited
```

For $sh$ or $ksh$, use:
```
$ ulimit -d unlimited
```

See the Solaris 64-bit Developer’s Guide for more information.
This chapter details the command-line options for the f95 compiler.

- A description of the syntax used for compiler option flags starts at “3.1 Command Syntax” on page 41.
- Summaries of options arranged by functionality starts at “3.3 Options Summary” on page 43.
- The complete reference detailing each compiler option flag starts at “3.4 Options Reference” on page 50.

3.1 Command Syntax

The general syntax of the compiler command line is:

```
f95 [options] list_of_files additional_options
```

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. Also, there are some options that must appear after the list of source files, and these could include additional lists of files (for example, -B, -l, and -L).

3.2 Options Syntax

Typical compiler option formats are:
3.2 Options Syntax

### TABLE 3–1  Options Syntax

<table>
<thead>
<tr>
<th>Syntax Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>–flag</td>
<td>–g</td>
</tr>
<tr>
<td>–flag=value</td>
<td>–Dnostep</td>
</tr>
<tr>
<td>–flag=value</td>
<td>–xunroll=4</td>
</tr>
<tr>
<td>–flag value</td>
<td>–o outfile</td>
</tr>
</tbody>
</table>

The following typographical conventions are used when describing the individual options:

### TABLE 3–2  Typographic Notations for Options

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
<th>Example: Text/Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>Square brackets contain arguments that are optional.</td>
<td>-O[n] -O4, -O</td>
</tr>
<tr>
<td>{}</td>
<td>Curly brackets (braces) contain a set of choices for a required option.</td>
<td>-d{y</td>
</tr>
<tr>
<td></td>
<td>The “pipe” or “bar” symbol separates arguments, only one of which may be chosen.</td>
<td>-B{dynamic</td>
</tr>
<tr>
<td>:</td>
<td>The colon, like the comma, is sometimes used to separate arguments.</td>
<td>-Rdir[:dir] -R/local/libs:/U/a</td>
</tr>
<tr>
<td>...</td>
<td>The ellipsis indicates omission in a series.</td>
<td>-xinline=f1[...fn] -xinline=alpha,dos</td>
</tr>
</tbody>
</table>

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). This 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.

- In an optional list of choices, such as `–xhasc=[yes|no]`, the first choice listed is the value assumed when the option flag appears on the command line without a value. For example, `–xhasc` is equivalent to `–xhasc=yes`. 
Source files, object files, and libraries are compiled and linked in the order in which they appear on the command line.

3.3 Options Summary

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.

Note that not all options are available on both SPARC and x64/x86 platforms. Check the detailed reference section for availability.

The following table summarizes the f95 compiler options by functionality. The table does not include obsolete and legacy option flags. Some flags serve more than one purpose and appear more than once.

<table>
<thead>
<tr>
<th>Function</th>
<th>Option Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compilation Mode:</strong></td>
<td></td>
</tr>
<tr>
<td>Compile only; do not produce an executable file</td>
<td>-c</td>
</tr>
<tr>
<td>Show commands built by the driver but do not compile</td>
<td>-dryrun</td>
</tr>
<tr>
<td>Support Fortran 77 extensions and compatibility</td>
<td>-f77</td>
</tr>
<tr>
<td>Specify path for writing compiled .mod Module files</td>
<td>-moddir=path</td>
</tr>
<tr>
<td>Specify name of object, library, or executable file to write</td>
<td>-o filename</td>
</tr>
<tr>
<td>Compile and generate only assembly code</td>
<td>-s</td>
</tr>
<tr>
<td>Strip symbol table from executable</td>
<td>-s</td>
</tr>
<tr>
<td>Suppress compiler messages, except error messages</td>
<td>-silent</td>
</tr>
<tr>
<td>Define path to directory for temporary files</td>
<td>-temp=path</td>
</tr>
<tr>
<td>Show elapsed time for each compilation phase</td>
<td>-time</td>
</tr>
<tr>
<td>Show version number of compiler and its phases</td>
<td>-V</td>
</tr>
<tr>
<td>Verbose messages</td>
<td>-v</td>
</tr>
<tr>
<td>Specify non-standard aliasing situations</td>
<td>-xalias=list</td>
</tr>
<tr>
<td>Compile with multiple processors</td>
<td>-xjobs=n</td>
</tr>
</tbody>
</table>

Compiled Code:
### TABLE 3-3  Compiler Options Grouped by Functionality  (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Option Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add/suppress trailing underscores on external names</td>
<td><code>-ext_names=x</code></td>
</tr>
<tr>
<td>Inline specified user functions</td>
<td><code>-inline=list</code></td>
</tr>
<tr>
<td>Compile position independent code</td>
<td><code>-KPIC/-kpic</code></td>
</tr>
<tr>
<td>Inline certain math library routines</td>
<td><code>-libmil</code></td>
</tr>
<tr>
<td>STOP returns integer status value to shell</td>
<td><code>-stop_status=[yn]</code></td>
</tr>
<tr>
<td>Specify code address space</td>
<td><code>-xcode=x</code></td>
</tr>
<tr>
<td>Enable prefetch instructions</td>
<td><code>-xprefetch=x</code></td>
</tr>
<tr>
<td>Specify use of optional registers</td>
<td><code>-xregs=x</code></td>
</tr>
<tr>
<td>Specify default data mappings</td>
<td><code>-xtypemap=x</code></td>
</tr>
</tbody>
</table>

**Data Alignment:**

| Specify alignment of data in COMMON blocks                               | `-aligncommon=[nN]` |
| Force COMMON block data alignment to allow double word fetch/store       | `-dalign`          |
| Force alignment of all data on 8-byte boundaries                          | `-dbl_align_all`   |
| Align COMMON block data on 8-byte boundaries                             | `-f`              |
| Specify memory alignment and behavior                                    | `-xmemalign=[ab]`  |

**Debugging:**

| Enable runtime subscript range checking                                  | `-C`              |
| Compile for debugging with `dbx`                                         | `-g`              |
| Flag use of undeclared variables                                         | `-u`              |
| Check `#PRAGMA ASSUME` assertions                                        | `-xassume_control=check` |
| Check for stack overflow at runtime                                      | `-xcheck=stkovf`  |
| Enable runtime task common check                                         | `-xcommonchk`     |
| Compile for Performance Analyzer                                         | `-xF`             |
| Generate cross-reference listings                                        | `-Xlist`          |
| Enable debugging without object files                                    | `-xs`             |

**Diagnostics:**

| Flag use of non-standard extensions                                      | `-ansi`           |
### TABLE 3-3 Compiler Options Grouped by Functionality (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Option Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suppress named warning messages</td>
<td>-erroff=</td>
</tr>
<tr>
<td>Display error tag names with error messages</td>
<td>-errtags</td>
</tr>
<tr>
<td>Show summary of compiler options</td>
<td>-flags, -help</td>
</tr>
<tr>
<td>Show version number of the compiler and its phases</td>
<td>-V</td>
</tr>
<tr>
<td>Verbose messages</td>
<td>-v</td>
</tr>
<tr>
<td>Verbose parallelization messages</td>
<td>-vpara</td>
</tr>
<tr>
<td>Show/suppress warning messages</td>
<td>-wr</td>
</tr>
<tr>
<td>Display compiler README file</td>
<td>-xhelp=readme</td>
</tr>
</tbody>
</table>

**Licensing:**

- Show license server information -xlicinfo

**Linking and Libraries:**

- Allow/require dynamic/static libraries -Bx
- Allow only dynamic/static library linking -dy, -dn
- Build a dynamic (shared object) library -G
- Assign name to dynamic library -bname
- Add directory to library search path -Lpath
- Link with library libname.a or libname.so -lname
- Do not build library search path into executable. -norunpath
- Build runtime library search path into executable -Rpath
- Disable use of incremental linker, ild -xildoff
- Link with optimized math library -xlibmopt
- Link with Sun Performance Library -xlic_lib=sunperf
- Link editor option -zx
- Generate pure libraries with no relocations -ztext

**Numerics and Floating-Point:**

- Use non-standard floating-point preferences -fnonstd
- Select SPARC non-standard floating point -fns
- Enable runtime floating-point overflow during input -fpover
### 3.3 Options Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Option Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select IEEE floating-point rounding mode</td>
<td>-fpround=r</td>
</tr>
<tr>
<td>Select floating-point optimization level</td>
<td>-fsimple=n</td>
</tr>
<tr>
<td>Select floating-point trapping mode</td>
<td>-ftrap=t</td>
</tr>
<tr>
<td>Specify rounding method for formatted input/output</td>
<td>-iorounding=mode</td>
</tr>
<tr>
<td>Promote single precision constants to double precision</td>
<td>-r8const</td>
</tr>
<tr>
<td>Enable interval arithmetic and set the appropriate floating-point environment (includes -xinterval)</td>
<td>-xia=[e]</td>
</tr>
<tr>
<td>Enable interval arithmetic extensions</td>
<td>-xinterval=[e]</td>
</tr>
</tbody>
</table>

**Optimization and Performance:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Option Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze loops for data dependencies</td>
<td>-depend</td>
</tr>
<tr>
<td>Optimize using a selection of options</td>
<td>-fast</td>
</tr>
<tr>
<td>Specify optimization level</td>
<td>-O [n]</td>
</tr>
<tr>
<td>Pad data layout for efficient use of cache</td>
<td>-pad=[p]</td>
</tr>
<tr>
<td>Allocate local variables on the memory stack</td>
<td>-stackvar</td>
</tr>
<tr>
<td>Enable loop unrolling</td>
<td>-unroll=[m]</td>
</tr>
<tr>
<td>Enable optimization across source files</td>
<td>-xcrossfile=[m]</td>
</tr>
<tr>
<td>Invoke interprocedural optimizations pass</td>
<td>-xipo=[m]</td>
</tr>
<tr>
<td>Set highest optimization level for #pragma OPT</td>
<td>-xmaxopt=[m]</td>
</tr>
<tr>
<td>Compile for post-compilation optimizations</td>
<td>-xbinopt=prepare</td>
</tr>
<tr>
<td>Enable/adjust compiler generated prefetch instructions</td>
<td>-xprefetch=[list]</td>
</tr>
<tr>
<td>Control automatic generation of prefetch instructions</td>
<td>-xprefetch_level=[n]</td>
</tr>
<tr>
<td>Enable generation or use of performance profiling data</td>
<td>-xprofile=[p]</td>
</tr>
<tr>
<td>Assert that no memory-based traps will occur</td>
<td>-xsafe=mem</td>
</tr>
<tr>
<td>Do no optimizations that increase code size</td>
<td>-xspace</td>
</tr>
<tr>
<td>Generate calls to vector library functions automatically</td>
<td>-xvector=[yn]</td>
</tr>
</tbody>
</table>

**Parallelization:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Option Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable automatic parallelization of DO loops</td>
<td>-autopar</td>
</tr>
<tr>
<td>Show loop parallelization information</td>
<td>-loopinfo</td>
</tr>
</tbody>
</table>
### TABLE 3–3  Compiler Options Grouped by Functionality  
(Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Option Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile for hand-coded multithreaded programming</td>
<td>-mt</td>
</tr>
<tr>
<td>Accept OpenMP API directives and set appropriate environment</td>
<td>-xopenmp[=keyword]</td>
</tr>
<tr>
<td>Recognize reduction operations in loops with automatic parallelization</td>
<td>-reduction</td>
</tr>
<tr>
<td>Verbose parallelization messages</td>
<td>-vpara</td>
</tr>
</tbody>
</table>

**Source Code:**

| Define preprocessor symbol | -Dname [=val] |
| Undefine preprocessor symbol | -Uname |
| Accept extended (132 character) source lines | -e |
| Apply preprocessor to .F and/or .F90 and .F95 files but do not compile | -F |
| Accept Fortran 95 fixed-format input | -fixed |
| Preprocess all source files with the fpp preprocessor | -fpp |
| Accept Fortran 95 free-format input | -free |
| Add directory to include file search path | -Ipath |
| Add directory to module search path | -Mpath |
| Recognize upper and lower case as distinct | -U |
| Tread hollerith as character in actual arguments | -xhasc={yes|no} |
| Select preprocessor, cpp or fpp, to use | -xpp={fpp|cpp} |
| Allow recursive subprogram calls | -xrecursive |

**Target Platform:**

| Specify memory model, 32 or 64 bits. | -m32 | -m64 |
| Specify target platform instruction set for the optimizer | -xarch= |
| Specify target cache properties for optimizer | -xcache= |
| Specify target processor for the optimizer | -xchip= |
| Specify target platform for the optimizer | -xtarget= |
### 3.3.1 Commonly Used Options

The compiler has 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 3-4** Commonly Used Options

<table>
<thead>
<tr>
<th>Action</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug—global program checking across routines for consistency of arguments, commons, and so on.</td>
<td>–Xlist</td>
</tr>
<tr>
<td>Debug—produce additional symbol table information to enable the dbx and debugging.</td>
<td>–g</td>
</tr>
<tr>
<td>Performance—invoke the optimizer to produce faster running programs.</td>
<td>–O[n]</td>
</tr>
<tr>
<td>Performance—Produce efficient compilation and run times for the native platform, using a set of predetermined options.</td>
<td>–fast</td>
</tr>
<tr>
<td>Dynamic (–Bdynamic) or static (–Bstatic) library binding.</td>
<td>–Bx</td>
</tr>
<tr>
<td>Compile only—Suppress linking; make a .o file for each source file.</td>
<td>–c</td>
</tr>
<tr>
<td>Output file—Name the executable output file nm instead of a.out.</td>
<td>–o nm</td>
</tr>
<tr>
<td>Source code—Compile fixed format Fortran source code.</td>
<td>-fixed</td>
</tr>
</tbody>
</table>

### 3.3.2 Macro Flags

Some option flags are macros that expand into a specific set of other flags. These are provided as a convenient way to specify a number of options that are usually expressed together to select a certain feature.

**TABLE 3-5** Macro Option Flags

<table>
<thead>
<tr>
<th>Option Flag</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dalign</td>
<td>-xmemalign=8s -aligncommon=16</td>
</tr>
<tr>
<td>-f</td>
<td>-aligncommon=16</td>
</tr>
<tr>
<td>-fast</td>
<td>See description of –fast for complete current expansion.</td>
</tr>
<tr>
<td>-fnonstd</td>
<td>-fns -ftrap=common</td>
</tr>
<tr>
<td>-xia=widestneed</td>
<td>-xinterval=widestneed -ftrap=%none -fns=no -fsimple=0</td>
</tr>
<tr>
<td>-xia=strict</td>
<td>-xinterval=strict -ftrap=%none -fns=no -fsimple=0</td>
</tr>
<tr>
<td>-xtarget</td>
<td>-xarch=a -xcache=b -xchip=c</td>
</tr>
</tbody>
</table>
Settings that follow the macro flag on the command line override or add to the the expansion of the macro.

### 3.3.3 Backward Compatibility and Legacy Options

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

**TABLE 3–6** Backward Compatibility Options

<table>
<thead>
<tr>
<th>Action</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preserve actual arguments over ENTRY statements</td>
<td>-arg=local</td>
</tr>
<tr>
<td>Allow assignment to constant arguments.</td>
<td>--copyargs</td>
</tr>
<tr>
<td>Treat hollerith constant as character or typeless in call argument lists</td>
<td>-xhasc=[{yes</td>
</tr>
<tr>
<td>Support Fortran 77 extensions and conventions</td>
<td>-f77</td>
</tr>
<tr>
<td>Nonstandard arithmetic—allow nonstandard arithmetic.</td>
<td>--fnonstd</td>
</tr>
<tr>
<td>Optimize performance for the host system.</td>
<td>--native</td>
</tr>
<tr>
<td>DO loops—use one trip DO loops.</td>
<td>--onetrip</td>
</tr>
<tr>
<td>Allow legacy aliasing situations</td>
<td>-xalias=keywords</td>
</tr>
</tbody>
</table>

Use of these option flags is not recommended for producing portable Fortran programs.

### 3.3.4 Obsolete Option Flags

The following options are considered obsolete and should not be used. They might be removed from later releases of the compiler.

**TABLE 3–7** Obsolete **f95** Options

<table>
<thead>
<tr>
<th>Option Flag</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>-xprofile=tcov</td>
</tr>
<tr>
<td>-cg89</td>
<td>-xtarget=ss2</td>
</tr>
<tr>
<td>-cg92</td>
<td>-xtarget=ss1000</td>
</tr>
</tbody>
</table>
### 3.4 Options Reference

This section describes all of the `f95` compiler command-line option flags, including various risks, restrictions, caveats, interactions, examples, and other details.

Unless indicated otherwise, each option is valid on both SPARC and x64/x86 platforms. Option flags valid only on SPARC platforms are marked (SPARC). Option flags valid only on x64/x86 platforms are marked (x86).

Option flags marked (Obsolete) are obsolete and should not be used. In many cases they have been superceded by other options or flags that should be used instead.

#### 3.4.1 –a

*(Obsolete)* Profile by basic block using `tcov`, old style.

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.
3.4.2 -aligncommon=[{1|2|4|8|16}]

Specify the alignment of data in common blocks and standard numeric sequence types.

The value indicates the maximum alignment (in bytes) for data elements within common blocks and standard numeric sequence types.

Note - A standard numeric sequence type is a derived type containing a SEQUENCE statement and only default component data types (INTEGER, REAL, DOUBLEPRECISION, COMPLEX without KIND= or * size). Any other type, such as REAL*8, will make the type non-standard.

For example, -aligncommon=4 would align data elements with natural alignments of 4 bytes or more on 4-byte boundaries.

This option does not affect data with natural alignment smaller than the specified size.

Without -aligncommon, the compiler aligns elements in common blocks and numeric sequence types on (at most) 4-byte boundaries.

Specifying -aligncommon without a value defaults to 1 - all common block and numeric sequence type elements align on byte boundaries (no padding between elements).

-aligncommon=16 reverts to -aligncommon=8 on platforms that are not 64-bit enabled.

Do not use -aligncommon=1 with -xmemalign as these declarations will conflict and could cause a segmentation fault on some platforms and configurations.

3.4.3 -ansi

Identify many nonstandard extensions.

Warning messages are issued for any uses of non–standard Fortran extensions in the source code.

3.4.4 -arg=local

Preserve actual arguments over ENTRY statements.

When you compile a subprogram with alternate entry points with this option, f95 uses copy/restore to preserve the association of dummy and actual arguments.

This option is provided for compatibility with legacy Fortran 77 programs. Code that relies on this option is non-standard.
3.4.5 –autopar

Enable automatic loop parallelization.

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.

Also specify the -stackvar option when using any of the parallelization options, including -autopar. The -stackvar option may provide better performance when using -autopar because it may allow the optimizer to detect additional opportunities for parallelization. See the description of the -stackvar option for information on how to set the sizes for the main thread stack and for the slave thread stacks.

Avoid -autopar if the program already contains explicit calls to the libthread threads library. See note in “3.4.56 –mt” on page 76.

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

To run a parallelized program in a multithreaded environment, you must set the PARALLEL (or OMP_NUM_THREADS) environment variable prior to execution. This tells the runtime system the maximum number of threads the program can create. The default is 1. In general, set the PARALLEL or OMP_NUM_THREADS variable to the number of available virtual processors on the target platform, which can be determined by using the Solaris psrinfo(1) command.

If you use -autopar and compile and link in one step, the multithreading library and the thread-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.

The-reduction option may also be useful with -autopar.

Refer to the Fortran Programming Guide for more information on parallelization. For explicit, user-controlled parallelization, use OpenMP directives and the -xopenmp option.

3.4.6 –B{static|dynamic}

Prefer dynamic or require static library linking.

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:

```bash
f95 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.

Mixing static Fortran runtime system libraries with dynamic Fortran runtime system libraries is not recommended and can result in linker errors or silent data corruption. Always link with the latest shared dynamic Fortran runtime system libraries.

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

---

### 3.4.7 -C

Check array references for out of range subscripts and conformance at runtime.

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. -C also adds runtime checks for array conformance in array syntax expressions.

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.

### 3.4.8 –c

Compile only; produce object .o files, but 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.

### 3.4.9 –copyargs

Allow assignment to constant arguments.

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.

### 3.4.10 –Dname[=def]

Define symbol name for the preprocessor.

This option only applies to .f, .f90, .f95, and .f03 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:

```bash
#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.
The predefined macro symbols have two leading underscores. The Fortran syntax may not support the actual values of these macros—they should appear only in `fpp` or `cpp` preprocessor directives. (Note the two leading underscores.)

- The product version is predefined (in hex) in `__SUNPRO_F90`, and `__SUNPRO_F95`. For example `__SUNPRO_F95` is `0x830` for the Sun Studio 12 release.

- The following macros are predefined on appropriate systems:
  - `__sparc`, `__unix`, `__sun`, `__SVR4`, `__i386`, `__SunOS_5_6`, `__SunOS_5_7`, `__SunOS_5_8`, `__SunOS_5_9`, `__SunOS_5_10`

  For instance, the value `__sparc` is defined on SPARC systems.

- The following are predefined with no underscores, but they might be deleted in a future release: `sparc`, `unix`, `sun`

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

- On 64-bit x86 systems, the macros `__amd64` and `__x86_64` are defined.

Compile with the verbose option (`-v`) to see the definitions created by the compiler.

You can use these values in such preprocessor conditionals as the following:

```
#define __sparc
```

`f95` uses 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` understands 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`.

### 3.4.11 –dalign

Align COMMON blocks and standard numerical sequence types, and generate faster multi-word load/stores.

This flag changes the data layout in COMMON blocks, numeric sequence types, 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 `-m64`). 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.

Using `-dalign` along with `-xtypemap=real:64,double:64,integer:64` also causes 64-bit integer variables to be double-word aligned on SPARC processors.
3.4 Options Reference

**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.

-`dalign` is a macro equivalent to:
  - `-xmemalign=8s -aligncommon=16` on SPARC platforms
  - `-aligncommon=8` on 32-bit x86 platforms
  - `-aligncommon=16` on 64-bit x86 platforms.

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

Note that because `-dalign` invokes `-aligncommon`, standard numeric sequence types are also affected by this option. See "3.4.2 –aligncommon\[={1|2|4|8|16}\]" on page 51

### 3.4.12 `-dbl_align_all[=\{yes|no\}]`

Force alignment of data on 8-byte boundaries

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

When compiling for 64-bit environments with `-m64`, 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.

Use with `-dalign` to enable added efficiency with multi-word load/stores.

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

### 3.4.13 `-depend[=\{yes|no\}]`

Analyze loops for inter-iteration data dependencies and performs loop restructuring. Loop restructuring includes loop interchange, loop fusion, scalar replacement, and elimination of "dead" array assignments.

On SPARC platforms, `-xdepend` is turned on for all optimization levels `-xO3` and above, and is off for lower opt levels. Also, an explicit setting of `-xdepend` overrides any implicit setting.

On x86 platforms, if optimization is not at `-xO3` or higher, the compiler raises the optimization to `-xO3` and issues a warning.
If you do not specify \texttt{-xdepend}, the default is \texttt{-xdepend=no} which means the compiler does not analyze loops for data dependencies. If you specify \texttt{-xdepend} but do not specify an argument, the compiler sets the option to \texttt{-xdepend=yes} which means the compiler analyzes loops for data dependencies.

Dependency analysis is included in \texttt{-xautotopar}. The dependency analysis is done at compile time. Dependency analysis may help on single-processor systems. However, if you try \texttt{-xdepend} on single-processor systems, you should not also specify \texttt{-xautotopar}, otherwise the \texttt{-xdepend} optimization is done for multiple-processor systems.

### 3.4.14 \texttt{--dn}

Disallow dynamic libraries. See “3.4.16 \texttt{--d(y|n)}” on page 57.

### 3.4.15 \texttt{--dryrun}

Show commands built by the \texttt{f95} command-line driver, but do not compile.

Useful when debugging, this option displays the commands and suboptions the compiler will invoke to perform the compilation.

### 3.4.16 \texttt{--d(y|n)}

Allow or disallow dynamic libraries for the entire executable.

- \texttt{--dy}: Yes, allow dynamic/shared libraries.
- \texttt{--dn}: No, do not allow dynamic/shared libraries.

The default, if not specified, is \texttt{--dy}.

Unlike \texttt{--Bx}, this option applies to the whole executable and need appear only once on the command line.

\texttt{--dy} | \texttt{--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 \texttt{libm.so} and \texttt{libc.so} (\texttt{libm.a} and \texttt{libc.a} are not provided). This means that \texttt{--dn} and \texttt{--Bstatic} may cause linking errors in 64-bit Solaris environments and 32-bit x86 Solaris platforms, and all 32-bit Solaris platforms starting with the Solaris 10 release. Applications must link with the dynamic libraries in these cases.

### 3.4.17 \texttt{--e}

Accept extended length input source line.
Extended source lines can be 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.

3.4.18  
-erroff[={%all|%none|taglist}]

Suppress warning messages listed by tag name.

Suppress the display of warning messages specified in the comma-separated list of tag names taglist. If %all, suppress all warnings, which is equivalent to the -w option. If %none, no warnings are suppressed. -erroff without an argument is equivalent to -erroff=%all.

Example:

f95 -erroff=WDECL_LOCAL_NOTUSED ink.f

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

3.4.19  
-errtags[={yes|no}]

Display the message tag with each warning message.

With -errtags=yes, the compiler’s internal error tag name will appear along with warning messages. -errtags alone is equivalent to -errtags=yes.

The default is not to display the tag (-errtags=no).

demo% f95 -errtags ink.f
ink.f:
MAIN:
"ink.f", line 11: Warning: local variable "i" never used (WDECL_LOCAL_NOTUSED)

3.4.20  
-errwarn[={%all|%none|taglist}]

Treat warning messages as errors.

The taglist specifies a list of comma-separated tag names of warning messages that should be treated as errors. If %all, treat all warnings as errors. If %none, no warnings are treated as errors.

See also -errtags.
### 3.4.21  
**–ext_names=e**

Create external names with or without trailing underscores.

*e* must be either **plain**, **underscores**, or **fsecond-underscore**. The default is **underscores**.

- **–ext_names=plain**: Do not add trailing underscore.
- **–ext_names=underscores**: Add trailing underscore.
- **–ext_names=fsecond-underscore**: Append two underscores to external names that contain an underscore, and a single underscore to those that do not.

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. Use this flag to allow Fortran routines to call (and be called by) other programming language routines.

**fsecond-underscore** is provided for compatibility with **gfortran**.

### 3.4.22  
**–F**

Invoke the source file preprocessor, but do not compile.

Apply the *fpp* preprocessor to *.F*, *.F90*, *.F95*, and *.F03* source files listed on the command line, and write the processed result on a file with the same name but with filename extension changed to *.f* (or *.f95* or *.f03*), but do not compile.

Example:

```
f95 -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**.

### 3.4.23  
**–f**

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

**–f** is a legacy option flag equivalent to **–aligncommon=16**. Use of **–aligncommon** is preferred.

The default alignment of data in COMMON blocks is on 4-byte boundaries. **–f** changes the data layout of double- and quad-precision data in COMMON blocks and EQUIVALENCE classes to be placed 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 SPARC environments with **–m64**).
Note – -f may result in nonstandard alignment of data, which could cause problems with variables in \texttt{EQUIVALENCE} or \texttt{COMMON} and may render the program non-portable if -f is required.

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 “3.4.11 –dalign” on page 55. Because -dalign is part of the -fast option, so is -f.

3.4.24 \texttt{-f77[=list]} \index{Fortran 77}

Select Fortran 77 compatibility mode.

This option flag enables porting legacy Fortran 77 source programs, including those with language extensions accepted by the \texttt{f77} compiler, to the \texttt{f95} Fortran compiler.

\texttt{list} is a comma-separated list selected from the following possible keywords:

\begin{tabular}{|l|l|}
\hline
\texttt{keyword} & \texttt{meaning} \\
\hline
\texttt{\%all} & Enable all the Fortran 77 compatibility features. \\
\hline
\texttt{\%none} & Disable all the Fortran 77 compatibility features. \\
\hline
\texttt{backslash} & Accept backslash as an escape sequence in character strings. \\
\hline
\texttt{input} & Allow input formats accepted by \texttt{f77}. \\
\hline
\texttt{intrinsics} & Limit recognition of intrinsics to only Fortran 77 intrinsics. \\
\hline
\texttt{logical} & Accept Fortran 77 usage of logical variables, such as: \\
& - assigning integer values to logical variables- allowing arithmetic expressions in logical conditional statements, with \texttt{.NE. 0} representing \texttt{.TRUE.}. - allowing relational operators \texttt{.EQ.} and \texttt{.NE.} with logical operands \\
\hline
\texttt{misc} & Allow miscellaneous \texttt{f77} Fortran 77 extensions. \\
\hline
\texttt{output} & Generate \texttt{f77}-style formatted output, including list-directed and \texttt{NAMELIST} output. \\
\hline
\texttt{subscript} & Allow non-integer expressions as array subscripts. \\
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>keyword</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab</td>
<td>Enable f77-style TAB-formatting, including unlimited source line length. No blank padding will be added to source lines shorter than 72 characters.</td>
</tr>
</tbody>
</table>

All keywords can be prefixed by no% to disable the feature, as in:

```
-f77=%all,no%backslash
```

The default, when -f77 is not specified, is -f77=%none. Using -f77 without a list is equivalent to specifying -f77=%all.

### Exceptions Trapping and -f77:

Specifying -f77 does not change the Fortran trapping mode, which is -ftrap=common. f95 differs from the Fortran 77 compiler’s behavior regarding arithmetic exception trapping. The Fortran 77 compiler allowed execution to continue after an arithmetic exception occurred. Compiling with -f77 also causes the program to call ieee_retrospective on program exit to report on any arithmetic exceptions that might have occurred. Specify -ftrap=%none following the -f77 option flag on the command line to mimic the original Fortran 77 behavior.

See "4.12 Mixing Languages" on page 166 for complete information on f77 compatibility and Fortran 77 to Fortran 95 migration.

See also the -xalias flag for handling non-standard programming syndromes that may cause incorrect results.

### 3.4.25 –fast

Select options that optimize execution performance.

#### 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 might not be available on all platforms. Compile with the -dryrun flag to see the expansion of -fast.

- **-fast** provides high performance for certain benchmark applications. However, the particular choice of options may or may not be appropriate for your application. Use -fast as a good starting point for compiling your application for best performance. But additional tuning may still be required. If your program behaves improperly when compiled with -fast, look closely at the individual options that make up -fast and invoke only those appropriate to your program that preserve correct behavior.

Note also that a program compiled with -fast may show good performance and accurate results with some data sets, but not with others. Avoid compiling with -fast those programs that depend on particular properties of floating-point arithmetic.
Because some of the options selected by -fast have linking implications, if you compile and link in separate steps be sure to link with -fast also.

-fast selects the following options:

- The -xtarget=native hardware target.
  - The program is intended to run on a different target than the compilation machine, follow the -fast with a code-generator option. For example: f95 -fast -xtarget=ultraT2 ...
- The -O5 optimization level option.
- The -depend option analyzes loops for data dependencies and possible restructuring.
- The -libmil option for system-supplied inline expansion templates. For C functions that depend on exception handling, follow -fast by -nolibmil (as in -fast -nolibmil). With -libmil, exceptions cannot be detected with errno or matherr(3m).
- The -fsimple=2 option for aggressive floating-point optimizations.
  - The -fsimple=2 is unsuitable if strict IEEE 754 standards compliance is required. See “3.4.36 -fsimple=[1|2|0]” on page 67.
- The -dalign option to generate double loads and stores for double and quad data in common blocks. Using this option can generate nonstandard Fortran data alignment in common blocks.
- The -xlibmopt option selects optimized math library routines.
- -pad=local inserts padding between local variables, where appropriate, to improve cache usage. (SPARC)
- -xvector=lib transforms certain math library calls within DO loops to single calls to a vectorized library equivalent routine with vector arguments. (SPARC)
- -fns selects non-standard floating-point arithmetic exception handling and gradual underflow. See “3.4.30 -fns=[yes|no]]” on page 64.
- -fround=nearest is selected because -xvector and -xlibmopt require it. (Solaris)
- Trapping on common floating-point exceptions, -ftrap=common, is enabled with f95.
- -nofstore cancels forcing expressions to have the precision of the result. (x86)
- -xregs=frameptr allows the compiler to use the frame-pointer register as an unallocated callee-saves register. Specify -xregs=no%frameptr after -fast and the frame pointer register will not be used as a general purpose register. (x86)

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

f95 -fast -fsimple=1 -xnolibmopt ...

which overrides the -fsimple=2 option and disables the -xlibmopt selected by -fast.
Because `-fast` invokes `-dalign`, `-fns`, `-fsimple=2`, programs compiled with `-fast` can result in nonstandard floating-point arithmetic, nonstandard alignment of data, and nonstandard ordering of expression evaluation. These selections might not be appropriate for most programs.

Note that the set of options selected by the `-fast` flag can change with each compiler release. Invoking the compiler with `-dryrun` displays the `-fast` expansion:

```bash
<sparc>%f95 -dryrun -fast |& grep ###
### command line files and options (expanded):
### -dryrun -xO5 -xarch=sparcvis2 -xcache=64/32/4:1024/64/4
   -xchip=ultra3i -xdepend=yes -xpad=local -xvector=lib
   -dalign -fsimple=2 -fns=yes -ftrap=common -xlibmil
   -xlibmopt -fround=nearest
```

### 3.4.26 –fixed

Specify fixed–format Fortran 95 source input files.

All source files on the command–line will be interpreted as fixed format regardless of filename extension. Normally, `f95` interprets only `.f` files as fixed format, `.f95` as free format.

### 3.4.27 –flags

Synonym for `-help`.

### 3.4.28 -fma={none|fused}

(SPARC) Enable automatic generation of floating-point, fused, multiply-add instructions. `-fma=none` disables generation of these instructions. `-fma=fused` allows the compiler to attempt to find opportunities to improve the performance of the code by using floating-point, fused, multiply-add instructions. The default is `-fma=none`.

Minimum requirements are `-xarch=sparcfma` and an optimization level of at least `-xO2` for the compiler to generate fused multiply-add instructions. The compiler will mark the binary program if fused multiply-add instructions have been generated to prevent execution of the program on platforms that do not support them.

Fused multiply-adds eliminate the intermediate rounding step between the multiply and the add. Consequently, programs may produce different results when compiled with `-fma=fused`, although precision will tend to be increased rather than decreased.


3.4 Options Reference

3.4.29  

**–fnonstd**

Initialize floating-point hardware to non-standard preferences.

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

**–fns -ftrap=common**

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

```fortran
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.
- 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*.

3.4.30  

**–fns [=yes|no]**

Select nonstandard floating-point mode.

The default is the 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** without a value is the same as **–fns=yes**.

This option flag enables nonstandard floating-point mode when the program begins execution. On SPARC platforms, specifying nonstandard floating-point mode disables "gradual underflow", causing tiny results to be flushed to zero rather than producing subnormal

Sun Studio 12 Update 1: Fortran User’s Guide •
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>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>( 0.0 &lt;</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>( 0.0 &lt;</td>
</tr>
</tbody>
</table>

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.

On x86 platforms, this option is enabled only for Pentium III and Pentium 4 processors (sse or sse2 instruction sets).

On x86, \(-fns\) selects SSE flush-to-zero mode and where available, denormals-are-zero mode. This flag causes subnormal results to be flushed to zero. Where available, this flag also causes subnormal operands to be treated as zero. This flag has no effect on traditional x87 floating-point operations not utilizing the SSE or SSE2 instruction set.

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

3.4.31 \(-fpo\varepsilonr\)[\(=\{\text{yes}\mid\text{no}\}]\)
Detect floating-point overflow in formatted input.

With \(-fpo\varepsilonrr=\text{yes}\) 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 (\(-fpo\varepsilonrr=\text{no}\)). \(-fpo\varepsilonrr\) without a value is equivalent to \(-fpo\varepsilonrr=\text{yes}\). Combine with \(-f\text{trap}\) to get full diagnostic information.

3.4.32 \(-f\text{pp}\)
Force preprocessing of input with \text{fpp}.
3.4 Options Reference

Pass all the input source files listed on the f95 command line through the fpp preprocessor, regardless of file extension. (Normally, only files with .F, .F90, or .F95 extension are automatically preprocessed by fpp.) See also "3.4.156 –xpp={fpp|cpp}" on page 126.

3.4.33 –fprecision={single|double|extended}

(x86) Initialize non-default floating-point rounding precision mode.

On x86, sets the floating-point precision mode to either single, double, or extended.

With a value of single or double, this flag causes the rounding precision mode to be set to single or double precision respectively at program initiation. With extended, or by default when the -fprecision flag is not specified, the rounding precision mode is initialized to extended precision.

This option is effective only on x86 systems and only if used when compiling the main program.

3.4.34 –free

Specify free–format source input files.

All source files on the command–line will be interpreted as f95 free format regardless of filename extension. Normally, f95 interprets .f files as fixed format, .f95 as free format.

3.4.35 –fround={nearest|tozero|negative|positive}

Set the IEEE rounding mode in effect at startup.

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 the value 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.)
3.4.36  \texttt{-fsimple[=\{1|2|0\}]}  
Select floating-point optimization preferences.

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 \texttt{-fsimple} option.

The defaults are:

- Without the \texttt{-fsimple} flag, the compiler defaults to \texttt{-fsimple=0}
- With \texttt{-fsimple} without a value, the compiler uses \texttt{-fsimple=1}

The different floating-point simplification levels are:

\texttt{-fsimple=0}  Permit no simplifying assumptions. Preserve strict IEEE 754 conformance.

\texttt{-fsimple=1}  Allow conservative simplifications. The resulting code does not strictly conform to IEEE 754, but numeric results of most programs are unchanged.

With \texttt{-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; for example, \texttt{x*0} may be replaced by \texttt{0}.
- Computations do not depend on sign of zero.

With \texttt{-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.

\texttt{-fsimple=2}  In addition to \texttt{-fsimple=1}, permit aggressive floating point optimizations. This can cause some programs to produce different numeric results due to changes in the way expressions are evaluated. In particular, the Fortran standard rule requiring compilers to honor explicit parentheses around subexpressions to control expression evaluation order may be broken with \texttt{-fsimple=2}. This could result in numerical rounding differences with programs that depend on this rule.

For example, with \texttt{-fsimple=2}, the compiler may evaluate \texttt{C-(A-B)} as \texttt{(C-A)+B}, breaking the standard’s rule about explicit parentheses, if the resulting code is better optimized. The compiler might also replace repeated
computations of \( x/y \) with \( x*z \), where \( z = 1/y \) is computed once and saved in a temporary, to eliminate the costly divide operations.

Programs that depend on particular properties of floating-point arithmetic should not be compiled with `-fsimple=2`.

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

`-fast` selects `-fsimple=2`.

### 3.4.37 `-fstore` *(x86)* Force precision of floating-point expressions.

For assignment statements, this option forces all floating-point expressions to the precision of the destination variable. This is the default. However, the `-fast` option includes `-nofstore` to disable this option. Follow `-fast` with `-fstore` to turn this option back on.

### 3.4.38 `-ftrap=t`

Set floating-point trapping mode in effect at startup.

`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`

`-ftrap=common` is a macro for `-ftrap=invalid,overflow,division`.

The `f95` default is `-ftrap=common`. This differs from the C and C++ compiler defaults, which is `-ftrap=none`.

Sets the IEEE 745 trapping mode in effect at startup but does not install a SIGFPE handler. You can use `ieee_handler(3M)` or `fex_set_handling(3M)` to simultaneously enable traps and install a SIGFPE handler. If you specify more than one value, the list is processed sequentially from left to right. The common exceptions, by definition, are invalid, division by zero, and 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, and will cause trapping of spurious and expected exceptions. Use `common` instead.
- `%none` 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*.

### 3.4.39 **–G**

Build a dynamic shared library instead of an executable file.

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.

### 3.4.40 **–g**

Compile for debugging and performance analysis.

Produce additional symbol table information for debugging with `dbx(1)` debugging utility and for performance analysis with the Performance Analyzer.

Although 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. See the `dbx` documentation for details.

To use the full capabilities of the Performance Analyzer, compile with `-g`. While some performance analysis features do not require `-g`, you must compile with `-g` to view annotated source, some function level information, and compiler commentary messages. (See the `analyzer(1)` man page and the manual *Sun Studio Performance Analyzer*.)

The commentary messages generated with `-g` describe the optimizations and transformations the compiler made while compiling your program. The messages, interleaved with the source code, can be displayed by the `er_src(1)` command.

Note that commentary messages only appear if the compiler actually performed any optimizations. You are more likely to see commentary messages when you request high optimization levels, such as with `-xO4`, or `-fast`. 
3.4.41 –hname

Specify the name of the generated dynamic shared library.

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

Without the -hname 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.

3.4.42 –help

Display a summary list of compiler options.

See also “3.4.125 –help={readme|flags}” on page 110.

3.4.43 –Ipath

Add path to the INCLUDE file search path.

Insert the directory path path at the start of the INCLUDE file search path. No space is allowed between -I and path. 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 f95 -I/usr/app/include growth.F

Multiple -Ipath 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 paths 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 compiler’s internal default list
4. /usr/include/

To invoke the preprocessor, you must be compiling source files with a .F, .F90, .F95, or .F03 suffix.

### 3.4.44 -i8

*(There is no i8 option.)*

Use -xtypemap=integer:64 to specify 8-byte INTEGER with this compiler.

### 3.4.45 -inline=%auto[[,][no%]f1,...[no%]fn]]

Enable or disable inlining of specified routines.

Request the optimizer to inline the user-written routines appearing in a comma-separated list of function and subroutine names. Prefixing a routine name with no% disables inlining of that routine.

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.

Specify %auto to enable automatic inlining at optimization levels -O4 or -O5. Automatic inlining at these optimization levels is normally turned off when explicit inlining is specified with -inline.

Example: Inline the routines xbar, zbar, vpoint:

```bash
demo% f95 -O3 -inline=xbar,zbar,vpoint *.f
```

Following are the restrictions; no warnings are issued:

- Optimization must be -O3 or greater.
- The source for the routine must be in the file being compiled, unless -xipo or -xcrossfile are also specified.
- The compiler determines if actual inlining is profitable and safe.
The appearance of -inline with -04 disables the automatic inlining that the compiler would normally perform, unless -auto is also specified. With -04, the compilers normally try to inline all appropriate user-written subroutines and functions. Adding -inline with -04 may degrade performance by restricting the optimizer’s inlining to only those routines in the list. In this case, use the %auto suboption to enable automatic inlining at -04 and -05.

```bash
demo f95 -O4 -inline=%auto,nozpoint *.f
```

In the example above, the user has enabled -04’s automatic inlining while disabling any possible inlining of the routine zpoint() that the compiler might attempt.

### 3.4.46 --iorounding[={compatible|processor-defined}]

Set floating-point rounding mode for formatted input/output.

Sets the ROUND= specifier globally for all formatted input/output operations.

With --iorounding=compatible, the value resulting from data conversion is the one closer to the two nearest representations, or the value away from zero if the value is halfway between them.

With --iorounding=processor-defined, the rounding mode is the processor’s default mode. This is the default when --iorounding is not specified.

### 3.4.47 --Kpic

*(Obsolete)* Synonym for --pic.

### 3.4.48 --KPIC

*(Obsolete)* Synonym for --PIC.

### 3.4.49 --L path

Add path to list of directory paths to search for libraries.

Adds path to the front of the list of object–library search directories. A space between -L and path is optional. This option is passed to the linker. See also "3.4.50 –Lx" on page 73.

While building the executable file, ld(1) searches path for archive libraries (.a files) and shared libraries (.so files). ld searches path 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 -L path, see ld(1).

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

Example: Use -L path to specify library search directories:

demo% f95 -L./dir1 -L./dir2 any.f

### 3.4.50 -lx

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

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 libVZY:

demo% f95 any.f -lvzy

Use -lx again to link with more libraries.

Example: Link with the libraries liby and libz:

demo% f95 any.f -ly -lz

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

### 3.4.51 -libmil

Inline selected libm library routines for optimization.

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.

For more information, see the man pages libm_single(3F) and libm_double(3F)
3.4.52  **–loopinfo**

Show loop parallelization results.

Show which loops were and were not parallelized with the **–autopar** option.

**–loopinfo** displays a list of messages on standard error:

```
demo% f95 -c -fast -autopar -loopinfo shalow.f
... 
'shalow.f', line 172: PARALLELIZED, and serial version generated
'shalow.f', line 173: not parallelized, not profitable
'shalow.f', line 181: PARALLELIZED, fused
'shalow.f', line 182: not parallelized, not profitable
... etc
```

3.4.53  **–Mpath**

Specify **MODULE** directory, archive, or file.

Look in path for Fortran modules referenced in the current compilation. This path is searched in addition to the current directory.

*path* can specify a directory, .a archive file of precompiled module files, or a .mod precompiled module file. The compiler determines the type of the file by examining its contents.

An archive .a file must be explicitly specified on a **–M** option flag to be searched for modules. The compiler will not search archive files by default.

Only .mod files with the same names as the **MODULE** names appearing on **USE** statements will be searched. For example, the statement **USE ME** causes the compiler to look only for the module file **me.mod**

When searching for modules, the compiler gives higher priority to the directory where the module files are being written. This is controlled by the **-moddir** compiler option, or the **MODDIR** environment variable. When neither are specified, the default write-directory is the current directory. When both are specified, the write-directory is the path specified by the **-moddir** flag.

This means that if only the **–M** flag appears, the current directory will be searched for modules first before any object listed on the **–M** flag. To emulate the behavior of previous releases, use:

```
-moddir=empty-dir -Mdir -M
```

where **empty-dir** is the path to an empty directory.
A space between the `-M` and the path is allowed. For example, `-M/home/siri/PK15/Modules`

On Solaris, if the path identifies a regular file that is not an archive or a module file, the compiler passes the option to the linker, `ld`, which will treat it as a linker map file. This feature is provided as a convenience similar to the C and C++ compilers.

See “4.9 Module Files” on page 163 for more information about modules in Fortran.

### 3.4.54 `–m32 | –m64`

Specify memory model for compiled binary object.

Use `–m32` to create 32-bit executables and shared libraries. Use `–m64` to create 64-bit executables and shared libraries.

The ILP32 memory model (32-bit int, long, pointer data types) is the default on all Solaris platforms and on Linux platforms that are not 64-bit enabled. The LP64 memory model (64-bit long, pointer data types) is the default on Linux platforms that are 64-bit enabled. `–m64` is permitted only on platforms that are enabled for the LP64 model.

Object files or libraries compiled with `–m32` cannot be linked with object files or libraries compiled with `–m64`.

When compiling applications with large amounts of static data using `–m64`, `-xmodel=medium` may also be required.

Be aware that some Linux platforms do not support the medium model.

Note that in previous compiler releases, the memory model, ILP32 or LP64, was implied by the choice of the instruction set with `–xarch`. Starting with the Sun Studio 12 compilers, this is no longer the case. On most platforms, just adding `–m64` to the command line is sufficient to create 64-bit objects.

On Solaris, `–m32` is the default. On Linux systems supporting 64-bit programs, `–m64 -xarch=sse2` is the default.

### 3.4.55 `–moddir=path`

Specify where the compiler will write compiled `.mod` MODULE files.

The compiler will write the `.mod` MODULE information files it compiles in the directory specified by `path`. The directory path can also be specified with the `MODDIR` environment variable. If both are specified, this option flag takes precedence.
The compiler uses the current directory as the default for writing .mod files.

See "4.9 Module Files" on page 163 for more information about modules in Fortran.

3.4.56 –mt

Require linking to thread-safe libraries.

If you do your own low-level thread management (for example, by calling the libthread library), compiling with -mt prevents conflicts.

Use -mt if you mix Fortran with multithreaded C code that calls the libthread library. See also the Solaris Multithreaded Programming Guide.

–mt is implied automatically when compiling with the -autopar option.

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 multithreaded code with -autopar. The compiler-generated calls to the threads library and the program’s own calls may conflict, causing unexpected results.
- On a single-processor system, performance may be degraded with the -mt option.

3.4.57 –native

(Obsolete) Optimize performance for the host system.

This option is a synonym for -xtarget=native, which is preferred. The -fast option sets -xtarget=native.

3.4.58 –noautopar

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

3.4.59 –nodepend

(SPARC) Cancel any -depend appearing earlier on the command line.
### 3.4.60 -nofstore

(x86) Cancel -fstore on command line.

The compiler default is -fstore. -fast includes -nofstore.

### 3.4.61 -nolib

Disable linking with system libraries.

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.

Link libm statically and libc dynamically with f95:

```bash
demo% f95 -nolib any.f95 -Bstatic -lm -Bdynamic -lc
```

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

### 3.4.62 -nolibmil

Cancel -libmil on command line.

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

```bash
demo% f95 -fast -nolibmil ...
```

### 3.4.63 -noreduction

Disable -reduction on command line.

This option disables -reduction.

### 3.4.64 -norunpath

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

## 3.4.65  
`-O[n]`

Specify optimization level.

`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 sets `-O3`) or `-fast` (which sets `-O5`) is recommended for most programs.

Each `-O[n]` 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 `-O[n]`, but `-O[n]` limits `-g` in certain ways; see the `dbx` documentation.

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.

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.”

## 3.4.66  
`-0`

This is equivalent to `-O3`. 

---

*Sun Studio 12 Update 1: Fortran User’s Guide*
3.4.67  
–O1
Provides a minimum of statement-level optimizations.
Use if higher levels result in excessive compilation time, or exceed available swap space.

3.4.68  
–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.

3.4.69  
–O3
Adds loop unrolling and global optimizations at the function level. Adds -depend automatically.
Usually -O3 generates larger executable files.

3.4.70  
–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.

3.4.71  
–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.
3.4.72 \texttt{-o name}

Specify the name of the executable file to be written.

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

3.4.73 \texttt{-onetrip}

Enable one trip \texttt{DO} loops.

Compile \texttt{DO} loops so that they are executed at least once. \texttt{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.

3.4.74 \texttt{-openmp}

Synonym for \texttt{-xopenmp}.

3.4.75 \texttt{-p}

\textit{(Obsolete)} Compile for profiling with the \texttt{prof} profiler.

Prepare object files for profiling, see \texttt{prof} (1). If you compile and link in separate steps, and also compile with the \texttt{-p} option, then be sure to link with the \texttt{-p} option. \texttt{-p} with \texttt{prof} is provided mostly for compatibility with older systems. \texttt{-pg} profiling with \texttt{gprof} is possibly a better alternative. See the \textit{Fortran Programming Guide} chapter on Performance Profiling for details.

3.4.76 \texttt{-pad[=p]}

Insert padding for efficient use of cache.

This option inserts padding between arrays or character variables, if they are static local and not initialized, or if they are 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 equivalence.

\textit{p}, if present, must be either \texttt{\%none} or either (or both) \texttt{local} or \texttt{common}. 


Add padding between adjacent local variables.

Add padding between variables in common blocks.

Do not add padding. (Compiler default.)

If both local and common are specified, they can appear in any order.

Defaults for -pad:

- The compiler does no padding by default.
- Specifying -pad, but without a value is equivalent to -pad=local,common.

The -pad 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 “3.4.91 –stackvar” on page 86.

The program must conform to the following restrictions:

- 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 with a warning message and the block is not padded.
- Avoid overindexing arrays in common blocks with -pad=common specified. The altered positioning of adjacent data in a padded common block will cause overindexing to fail in unpredictable ways.

It is the programmer’s responsibility to make sure that common blocks are compiled consistently when -pad is used. Common blocks appearing in different program units that are compiled inconsistently with -pad=common will cause errors. Compiling with -Xlist will report when common blocks with the same name have different lengths in different program units.
3.4.77  

-pg

Compile for profiling with the gprof profiler.

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 source and .o files ( -pg libraries are static).

Note - There is no advantage compiling with -xprofile if you specify -pg. These two features do not prepare or use data provided by the other.

Profiles generated by using prof(1) or gprof(1) on 64 bit Solaris platforms or just gprof on 32 bit Solaris platforms include approximate user CPU times. These times are derived from PC sample data (see pcsample(2)) for routines in the main executable and routines in shared libraries specified as linker arguments when the executable is linked. Other shared libraries (libraries opened after process startup using dlopen(3DL)) are not profiled.

On 32 bit Solaris systems, profiles generated using prof(1) are limited to routines in the executable. 32 bit shared libraries can be profiled by linking the executable with -pg and using gprof(1).

The Solaris 10 software does not include system libraries compiled with -p. As a result, profiles collected on Solaris 10 platforms do not include call counts for system library routines.

The compiler options -p, -pg, or -xpg should not be used to compile multi-threaded programs, because the runtime support for these options is not thread-safe. If a program that uses multiple threads is compiled with these options invalid results or a segmentation fault could occur at runtime.

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

3.4.78  

-pic

Compile position-independent code for shared library.

On SPARC, -pic is equivalent to -xcode=pic13. See "3.4.117 -xcode=keyword" on page 104 for more information on position-independent code.

On x86, produces position-independent code. Use this option to compile source files when building a shared library. 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 pc-relative addressing mode through a procedure linkage table.
3.4.79 –PIC

Compile position–independent code with 32-bit addresses.

On SPARC, –PIC is equivalent to -xcode=pic32. See “3.4.117 –xcode=keyword” on page 104 for more information about position-independent code.

On x86, –PIC is equivalent to –pic.

3.4.80 –Qoption pr ls

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.

3.4.81 –qp

Synonym for -p.

3.4.82 –R ls

Build dynamic library search paths into the executable file.

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.

3.4.83 **–r8const**

Promote single-precision constants to REAL*8 constants.

All single-precision REAL constants are promoted to REAL*8. Double-precision (REAL*8) constants are not changed. This option only applies to constants. To promote both constants and variables, see “3.4.173 –xtypemap=spec” on page 137.

Use this option flag carefully. It could cause interface problems when a subroutine or function expecting a REAL*4 argument is called with a REAL*4 constant that gets promoted to REAL*8. It could also cause problems with programs reading unformatted data files written by an unformatted write with REAL*4 constants on the I/O list.

3.4.84 **-recl=a[,b]**

Set default output record length.

Set the default record length (in characters) for either or both preconnected units output (standard output) and error (standard error). This option must be specified using one of the following forms:

- `-recl=out:N`
- `-recl=error:N`
- `-recl=out:N1,error:N2`
- `-recl=error:N1,out:N2`
- `-recl=all:N`

where N, N1, N2 are all positive integers in the range from 72 to 2147483646. **out** refers to standard output, **error** to standard error, and **all** sets the default record length to both. The default is `-recl=all:80`. This option is only effective if the program being compiled has a Fortran main program.

3.4.85 **–reduction**

Recognize reduction operations in loops.

Analyze loops for reduction operations during automatic parallelization. There is potential for roundoff error with the reduction.
A reduction operation accumulates the elements of an array into a single scalar value. 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 is usable only with the automatic parallelization option `-autopar`. It is ignored otherwise. Explicitly parallelized loops are not analyzed for reduction operations.

### 3.4.86 –S

Compile and only generate assembly code.

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

### 3.4.87 –s

Strip the symbol table out of the executable file.

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`.

### 3.4.88 –sb

*(Obsolete—this option is ignored)*

### 3.4.89 –sbfast

*(Obsolete — this option is ignored.)*

### 3.4.90 –silent

*(Obsolete) Suppress compiler messages.*

Normally, the *f95* compiler does not issue messages, other than error diagnostics, during compilation. This option flag is provided for compatibility with the legacy *f77* compiler, and its use is redundant except with the `-f77` compatibility flag.
3.4 Options Reference

3.4.91 –stackvar

Allocate local variables on the stack whenever possible.

This option makes writing recursive and re-entrant code easier and provides the optimizer more freedom when parallelizing loops.

Use of –stackvar is recommended with any of the parallelization options.

Local variables are variables that are not dummy arguments, COMMON variables, variables inherited from an outer scope, or module variables made accessible by a USE statement.

With –stackvar in effect, local variables are allocated on the stack unless they have the attributes SAVE or STATIC. Note that explicitly initialized variables are implicitly declared with the SAVE attribute. A structure variable that is not explicitly initialized but some of whose components are initialized is, by default, not implicitly declared SAVE. Also, variables equivalenced with variables that have the SAVE or STATIC attribute are implicitly SAVE or STATIC.

A statically allocated variable is implicitly initialized to zero unless the program explicitly specifies an initial value for it. Variables allocated on the stack are not implicitly initialized except that components of structure variables can be initialized by default.

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

The initial thread executing the program has a main stack, while each helper thread of a multithreaded program has its own thread stack.

The default size of the main stack is about 8 Megabytes. The default thread stack size is 4 Megabytes on 32-bit systems, 8 Megabytes on 64-bit systems. The limit command (with no parameters) shows the current main stack size. If you get a segmentation fault using –stackvar, try increasing the main and thread stack sizes.

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
```

You can set the stack size to be used by each slave thread by giving the `STACKSIZE` environment variable a value (in Kilobytes):

```
% setenv STACKSIZE 8192
```

sets the stack size for each slave thread to 8 Mb.

The `STACKSIZE` environment variable also accepts numerical values with a suffix of either `B`, `K`, `M`, or `G` for bytes, kilobytes, megabytes, or gigabytes respectively. The default is kilobytes.

Note that the `STACKSIZE` environment variable affects only programs compiled with the `-xopenmp` or `-xautopar` options, and has no affect on programs using the pthreads interface on Solaris systems.

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

Compile with `-xcheck=stkovf` to enable runtime checking for stack overflow situations. See "3.4.115 –xcheck=keyword" on page 102.

### 3.4.92 `–stop_status=#{yes|no}`

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

The default is `-stop_status=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`.

### 3.4.93 `–temp=dir`

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

This option takes precedence over the value of the \texttt{TMPDIR} environment variable.

3.4.94 \texttt{--time}

Time each compilation phase.

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

3.4.95 \texttt{--U}

Recognize upper and lower case in source files.

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 \texttt{Delta}, \texttt{DELTA}, and \texttt{delta} as different symbols. Calls to intrinsic functions are not affected by this option.

Portability and mixing Fortran with other languages may require use of \texttt{--U}. See the Fortran Programming Guide chapter on porting programs to Sun Studio Fortran.

3.4.96 \texttt{--Uname}

Undefine preprocessor macro \texttt{name}.

This option applies only to source files that invoke the \texttt{fpp} or \texttt{cpp} pre-processor. It removes any initial definition of the preprocessor macro \texttt{name} created by \texttt{--Dname} on the same command line, including those implicitly placed there by the command-line driver, regardless of the order the options appear. It has no effect on any macro definitions in source files. Multiple \texttt{--Uname} flags can appear on the command line. There must be no space between \texttt{--U} and the macro \texttt{name}.

3.4.97 \texttt{--u}

Report undeclared variables.

Make the default type for all variables be \texttt{undeclared} rather than using Fortran implicit typing, as if \texttt{IMPLICIT NONE} appeared in each compilation unit. This option warns of undeclared variables, and does not override any \texttt{IMPLICIT} statements or explicit \texttt{type} statements.
3.4.98  

--unroll=n

Enable unrolling of DO loops where possible.

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 “2.3.1.3 The UNROLL Directive” on page 32.

3.4.99  

--use=list

Specify implicit USE modules.

list is a comma-separated list of module names or module file names.

Compiling with -use=module_name has the effect of adding a USE module_name statement to each subprogram or module being compiled. Compiling with -use=module_file_name has the effect of adding a USE module_name for each of the modules contained in the specified file.

See “4.9 Module Files” on page 163 for more information about modules in Fortran.

3.4.100  

-V

Show name and version of each compiler pass.

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

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

3.4.101  

-v

Verbose mode - show details of each compiler pass.

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

3.4.102  

-vax=keywords

Specify choice of VAX VMS Fortran extensions enabled.
The *keywords* specifier must be one of the following suboptions or a comma-delimited list of a selection of these.

<table>
<thead>
<tr>
<th>Sub-option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>blank_zero</strong></td>
<td>Interpret blanks in formatted input as zeros on internal files.</td>
</tr>
<tr>
<td><strong>debug</strong></td>
<td>Interpret lines starting with the character 'D' to be regular Fortran statements rather than comments, as in VMS Fortran.</td>
</tr>
<tr>
<td><strong>rsize</strong></td>
<td>Interpret unformatted record size to be in words rather than bytes.</td>
</tr>
<tr>
<td><strong>struct_align</strong></td>
<td>Layout components of a VAX structure in memory as in VMS Fortran, without padding. Note: this can cause data misalignments, and should be used with --xmemalign to avoid such errors.</td>
</tr>
<tr>
<td><strong>%all</strong></td>
<td>Enable all these VAX VMS features.</td>
</tr>
<tr>
<td><strong>%none</strong></td>
<td>Disable all these VAX VMS features.</td>
</tr>
</tbody>
</table>

Sub-options can be individually selected or turned off by preceeding with no%.

Example:

```
-vax=debug,rsize,no%blank_zero
```

### 3.4.103 \(-\text{vpara}\)

Show verbose parallelization messages.

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

Use with -xopenmp and OpenMP API directives.

Warnings are issued with the compiler detects the following situations:

- Problematic use of OpenMP data sharing attributes clauses, such as declaring a variable shared whose accesses in an OpenMP parallel region may cause data race, or declaring a variable private whose value in a parallel region is used after the parallel region.

No warnings appear if all parallelization directives are processed without issues.

**Note** – Sun Studio compilers support the OpenMP API parallelization model. Consequently, legacy $C$MIC parallelization directives are deprecated and ignored. See the *OpenMP API User’s Guide* for information on migrating to the OpenMP API.
### 3.4.104 \(-w[n]\)

Show or suppress warning messages.

This option shows or 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.

\(n\) may be 0, 1, 2, 3, or 4.

\(-w0\) shows just error messages. This is equivalent to \(-w\). \(-w1\) shows errors and warnings. This is the default without \(-w\). \(-w2\) shows errors, warnings, and cautions. \(-w3\) shows errors, warnings, cautions, and notes. \(-w4\) shows errors, warnings, cautions, notes, and comments.

### 3.4.105 \(-Xlist[x]\)

*(Solaris only)* Produce listings and do global program checking (GPC).

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 \(-Xlist\). Unpredictable reports may result when run on a source code with syntax errors.

Example: Check across routines for consistency:

```bash
demo% f95 -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
### 3.4 Options Reference

#### 3.4.106 \(-xa\)

Synonym for \(-a\).

#### 3.4.107 \(-xaddr32[={yes|no}]\)

(x86/x64 only) The \(-xaddr32=yes\) compilation flag restricts the resulting executable or shared object to a 32-bit address space.

---

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

This option is not available on Linux systems.
An executable that is compiled in this manner results in the creation of a process that is restricted to a 32-bit address space. When \texttt{-xaddr32=no} is specified a usual 64 bit binary is produced. If the \texttt{-xaddr32} option is not specified, \texttt{-xaddr32=no} is assumed. If only \texttt{-xaddr32} is specified \texttt{-xaddr32=yes} is assumed.

This option is only applicable to \texttt{-mm64} compilations and only on Solaris platforms supporting \texttt{SF1\_SUNW\_ADDR32} software capability. Since Linux kernel does not support address space limitation this option is not available on Linux. The \texttt{-xaddr32} option is ignored on Linux.

When linking, if a single object file was compiled with \texttt{-xaddr32=yes} the whole output file is assumed to be compiled with \texttt{-xaddr32=yes}. A shared object that is restricted to a 32-bit address space must be loaded by a process that executes within a restricted 32-bit mode address space. For more information refer to the \texttt{SF1\_SUNW\_ADDR32} software capabilities definition, described in the \textit{Linker and Libraries Guide}.

### 3.4.108 \texttt{-xalias[=keywords]}

Specify degree of aliasing to be assumed by the compiler.

Some non-standard programming techniques can introduce situations that interfere with the compiler’s optimization strategies. The use of overindexing, pointers, and passing global or non-unique variables as subprogram arguments, can introduce ambiguous aliasing situations that could result code that does not work as expected.

Use the \texttt{-xalias} flag to inform the compiler about the degree to which the program deviates from the aliasing requirements of the Fortran standard.

The flag may appear with or without a list of keywords. The \textit{keywords} list is comma-separated, and each keyword indicates an aliasing situation present in the program.

Each keyword may be prefixed by \texttt{no\%} to indicate an aliasing type that is not present.

The aliasing keywords are:

<table>
<thead>
<tr>
<th>keyword</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>dummy</td>
<td>Dummy (formal) subprogram parameters can alias each other and global variables.</td>
</tr>
<tr>
<td>no%dummy</td>
<td>(Default). Usage of dummy parameters follows the Fortran standard and do not alias each other or global variables.</td>
</tr>
</tbody>
</table>
TABLE 3-10  -xalias Option Keywords  (Continued)

<table>
<thead>
<tr>
<th>keyword</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>craypointer</td>
<td>(Default). Cray pointers can point at any global variable or a local variable whose address is taken by the LOC() function. Also, two Cray pointers might point at the same data. This is a safe assumption that could inhibit some optimizations.</td>
</tr>
<tr>
<td>no%craypointer</td>
<td>Cray pointers point only at unique memory addresses, such as obtained from malloc(). Also, no two Cray pointers point at the same data. This assumption enables the compiler to optimize Cray pointer references.</td>
</tr>
<tr>
<td>actual</td>
<td>The compiler treats actual subprogram arguments as if they were global variables. Passing an argument to a subprogram might result in aliasing through Cray pointers.</td>
</tr>
<tr>
<td>no%actual</td>
<td>(Default) Passing an argument does not result in further aliasing.</td>
</tr>
<tr>
<td>overindex</td>
<td>■ A reference to an element in a COMMON block might refer to any element in a COMMON block or equivalence group.</td>
</tr>
<tr>
<td></td>
<td>■ Passing any element of a COMMON block or equivalence group as an actual argument to a subprogram gives access to any element of that COMMON block or equivalence group to the called subprogram.</td>
</tr>
<tr>
<td></td>
<td>■ Variables of a sequence derived type are treated as if they were COMMON blocks, and elements of such a variable might alias other elements of that variable.</td>
</tr>
<tr>
<td></td>
<td>■ Individual array bounds may be violated, but except as noted above, the referenced array element is assumed to stay within the array. Array syntax, WHERE, and FORALL statements are not considered for overindexing. If overindexing occurs in these constructs, they should be rewritten as DO loops.</td>
</tr>
<tr>
<td>no%overindex</td>
<td>(Default) Array bounds are not violated. Array references do not reference other variables.</td>
</tr>
<tr>
<td>ftnpointer</td>
<td>Calls to external functions might cause Fortran pointers to point at target variables of any type, kind, or rank.</td>
</tr>
<tr>
<td>no%ftnpointer</td>
<td>(Default) Fortran pointers follow the rules of the standard.</td>
</tr>
</tbody>
</table>

Specifying -xalias without a list gives the best performance for most programs that do not violate Fortran aliasing rules, and corresponds to:

no%dummy, no%craypointer, no%actual, no%overindex, no%ftnpointer

To be effective, -xalias should be used when compiling with optimization levels -xO3 and higher.

The compiler default, with no -xalias flag specified, assumes that the program conforms to the Fortran standard except for Cray pointers:
Examples of various aliasing situations and how to specify them with `-xalias` are given in the Porting chapter of the *Fortran Programming Guide*.

### 3.4.109 `-xannotate=[{yes|no}]`

(Solaris Only) Instructs the compiler to create binaries that can later be transformed by binary modification tools like `binopt(1)`.

Future binary analysis, code coverage and memory error detection tools will also work with binaries built with this option.

Use the `-xannotate=no` option to prevent the modification of the binary file by these tools. The `-xannotate=yes` option must be used with optimization level `-xO1` or higher to be effective, and it is only effective on systems with the new linker support library interface `-ld_open()`. If the compiler is used on a system without this linker interface (for example Solaris 9 and early versions of Solaris 10), it silently will revert to `-xannotate=no`.

The default is `-xannotate=yes`, but if either of the above conditions is not met, the default reverts to `-xannotate=no`.

This option is not available on Linux systems.

### 3.4.110 `–xarch=isa`

Specify instruction set architecture (ISA).

Architectures that are accepted by `-xarch` keyword `isa` are shown in Table 3–11:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Valid <code>-xarch</code> Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARC</td>
<td><code>generic, generic64, native, native64, sparc, sparcvis, sparcvis2, sparcfmaf, v9, v9a, v9b</code></td>
</tr>
<tr>
<td>x86</td>
<td><code>generic, native, 386, pentium_pro, sse, sse2, amd64, pentium_proa, ssea, sse2a, amd64a, sse3, sse3a, ssse3, sse4_1, sse4_2, amdsse4a</code></td>
</tr>
</tbody>
</table>

Note that although `-xarch` can be used alone, it is part of the expansion of the `–xtarget` option and may be used to override the `-xarch` value that is set by a specific `–xtarget` option. For example:

```
% f95 -xtarget=ultra2 -xarch=sparcfmaf ...
```
overrides the -xarch set by -xtarget=ultra2

This option limits the code generated by the compiler to the instructions of the specified instruction set architecture by allowing only the specified set of instructions. This option does not guarantee use of any target-specific instructions.

If this option is used with optimization, the appropriate choice can provide good performance of the executable on the specified architecture. An inappropriate choice results in a binary program that is not executable on the intended target platform.

Note the following:
- Legacy 32-bit SPARC instruction set architectures V7 and V8 imply -m32 and cannot be combined with -m64.
- Object binary files (.o) compiled with sparc and sparcvis can be linked and can execute together, but will run only on a sparcvis—compatible platform.
- Object binary files (.o) compiled with sparc, sparcvis, and sparcvis2 can be linked and can execute together, but will run only on a sparcvis2—compatible platform.

For any particular choice, the generated executable may run much more slowly on earlier architectures. Also, although quad-precision (REAL*16 and long double) floating-point instructions are available in many of these instruction set architectures, the compiler does not use these instructions in the code it generates.

The default when -xarch is not specified is generic.

Table 3–12 gives details for each of the -xarch keywords on SPARC platforms.

<table>
<thead>
<tr>
<th>-xarch</th>
<th>Meaning (SPARC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>Compile using the instruction set common to most processors. This is v8plus when compiling with -m32, and sparc with -m64.</td>
</tr>
<tr>
<td>generic64</td>
<td>Compile for most 64-bit platforms. (Solaris only) This option is equivalent to -m64 -xarch=generic and is provided for compatibility with earlier releases. Use -m64 to specify 64-bit compilation instead of -xarch=generic64</td>
</tr>
<tr>
<td>native</td>
<td>Compile for good performance on this system. The compiler chooses the appropriate setting for the current system processor it is running on. This is the default for the -fast option.</td>
</tr>
</tbody>
</table>
### TABLE 3.12 -xarch Values for SPARC Platforms  (Continued)

<table>
<thead>
<tr>
<th>-xarch=</th>
<th>Meaning (SPARC)</th>
</tr>
</thead>
</table>
| native64 | Compile for good performance in 64-bit mode on this system.  
(Solaris only) This option is equivalent to -m64 -xarch=native and is provided for compatibility with earlier releases. |
| sparc | Compile for the SPARC-V9 ISA.  
Compile for the V9 ISA, but without the Visual Instruction Set (VIS), and without other implementation-specific ISA extensions. This option enables the compiler to generate code for good performance on the V9 ISA. |
| sparcvis | Compile for the SPARC-V9 ISA with UltraSPARC extensions.  
Compile for SPARC-V9 plus the Visual Instruction Set (VIS) version 1.0, and with UltraSPARC extensions. This option enables the compiler to generate code for good performance on the UltraSPARC architecture. |
| sparcvis2 | Compile for the SPARC-V9 ISA with UltraSPARC-III extensions.  
Enables the compiler to generate object code for the UltraSPARC architecture, plus the Visual Instruction Set (VIS) version 2.0, and with UltraSPARC-III extensions. |
| sparcfmaf | Compile for the sparcfmaf version of the SPARC-V9 ISA.  
Enables the compiler to use instructions from the SPARC-V9 instruction set, plus the UltraSPARC extensions, including the Visual Instruction Set (VIS) version 1.0, the UltraSPARC-III extensions, including the Visual Instruction Set (VIS) version 2.0, and the SPARC64 V1 extensions for floating-point multiply-add.  
Note that you must use -xarch=sparcfmaf in conjunction with -fma=fused and some optimization level to get the compiler to attempt to find opportunities to use the multiply-add instructions automatically. |
| v9 | Equivalent to -m64 -xarch=sparc  
Legacy makefiles and scripts that use -xarch=v9 to obtain the 64-bit memory model need only use -m64. |
| v9a | Equivalent to -m64 -xarch=sparcvis and is provided for compatibility with earlier releases. |
| v9b | Equivalent to -m64 -xarch=sparcvis2 and is provided for compatibility with earlier releases. |

Table 3–13 details each of the -xarch keywords on x86 platforms. The default on x86 is generic (or generic64 if -m64 is specified) if -xarch is not specified.
### Table 3-13: -xarch Values for x86 Platforms

<table>
<thead>
<tr>
<th>-xarch=</th>
<th>Meaning (x86)</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>Compile for good performance on most 32-bit x86 platforms. This is the default, and is equivalent to -xarch=pentium_pro.</td>
</tr>
<tr>
<td>generic64</td>
<td>Compile for good performance on most 64-bit x86 platforms. It is equivalent to sse2.</td>
</tr>
<tr>
<td>native</td>
<td>Compile for good performance on this x86 architecture. Use the best instruction set for good performance on most x86 processors. With each new release, the definition of &quot;best&quot; instruction set may be adjusted, if appropriate.</td>
</tr>
<tr>
<td>native64</td>
<td>Compile for good performance on this 64-bit x86 architecture.</td>
</tr>
<tr>
<td>386</td>
<td>Limits instruction set to the Intel 386/486 architecture.</td>
</tr>
<tr>
<td>pentium_pro</td>
<td>Limits instruction set to the Pentium Pro architecture.</td>
</tr>
<tr>
<td>pentium_proa</td>
<td>Adds the AMD extensions (3DNow!, 3DNow! extensions, and MMX extensions) to the 32-bit Pentium Pro architecture.</td>
</tr>
<tr>
<td>sse</td>
<td>Adds the SSE instruction set to pentium_pro. (See Note below.)</td>
</tr>
<tr>
<td>ssea</td>
<td>Adds the AMD extensions (3DNow!, 3DNow! extensions, and MMX extensions) to the 32-bit SSE architecture.</td>
</tr>
<tr>
<td>sse2</td>
<td>Adds the SSE2 instruction set to the pentium_pro. (See Note below.)</td>
</tr>
<tr>
<td>sse2a</td>
<td>Adds the AMD extensions (3DNow!, 3DNow! extensions, and MMX extensions) to the 32-bit SSE2 architecture.</td>
</tr>
<tr>
<td>sse3</td>
<td>Adds the SSE3 instruction set to the SSE2 instruction set.</td>
</tr>
<tr>
<td>amd64</td>
<td>On Solaris platforms, this is equivalent to -m64 -xarch=sse2. Legacy makefiles and scripts that use -xarch=amd64 to obtain the 64-bit memory model should use -m64.</td>
</tr>
<tr>
<td>amd64a</td>
<td>On Solaris platforms, this is equivalent to -m64 -xarch=sse2a.</td>
</tr>
<tr>
<td>sse3a</td>
<td>Adds AMD extended instructions, including 3DNow! to the SSE3 instruction set.</td>
</tr>
<tr>
<td>ssse3</td>
<td>Adds the SSSE3 instructions to the SSE3 instruction set.</td>
</tr>
<tr>
<td>sse4_1</td>
<td>Adds the SSE4.1 instructions to the SSSE3 instruction set.</td>
</tr>
<tr>
<td>sse4_2</td>
<td>Adds the SSE4.2 instructions to the SSE4.1 instruction set.</td>
</tr>
<tr>
<td>amdsse4a</td>
<td>Adds the SSE4a instructions to the AMD instruction set.</td>
</tr>
</tbody>
</table>

#### 3.4.110.1 Special Cautions for x86/x64 Platforms:
There are some important considerations when compiling for x86 Solaris platforms.
-xarch set to sse, sse2, sse2a, or sse3 and beyond must be run on platforms supporting these features and extensions.

OS releases starting with Solaris 9 4/04 are SSE/SSE2-enabled on Pentium 4-compatible platforms. Earlier versions of Solaris OS are not SSE/SSE2-enabled.

If you compile and link in separate steps, always link using the compiler and with same -xarch setting to ensure that the correct startup routine is linked.

Arithmetic results on x86 may differ from results on SPARC due to the x86 80-byte floating-point registers. To minimize these differences, use the -fstore option or compile with -xarch=sse2 if the hardware supports SSE2.

Starting with Sun Studio 11 and the Solaris 10 OS, program binaries compiled and built using these specialized -xarch hardware flags are verified that they are being run on the appropriate platform.

On systems prior to Solaris 10, no verification is done and it is the user’s responsibility to ensure objects built using these flags are deployed on suitable hardware.

Running programs compiled with these -xarch options on platforms that are not enabled with the appropriate features or instruction set extensions could result in segmentation faults or incorrect results occurring without any explicit warning messages.

This warning extends also to programs that employ .il inline assembly language functions or __asm() assembler code that utilize SSE, SSE2, SSE2a, and SSE3 instructions and extensions.

3.4.111 -xassume_control[=keywords]

Set parameters to control ASSUME pragmas.

Use this flag to control the way the compiler handles ASSUME pragmas in the source code.

The ASSUME pragmas provide a way for the programmer to assert special information that the compiler can use for better optimization. These assertions may be qualified with a probability value. Those with a probability of 0 or 1 are marked as certain; otherwise they are considered non-certain.

You can also assert, with a probability or certainty, the trip count of an upcoming DO loop, or that an upcoming branch will be taken.

See “2.3.1.8 The ASSUME Directives” on page 34, for a description of the ASSUME pragmas recognized by the f95 compiler.

The keywords on the -xassume_control option can be a single suboption keyword or a comma-separated list of keywords. The keyword suboptions recognized are:
The assertions made on **ASSUME** pragmas affect optimization of the program.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>optimize</strong></td>
<td>The assertions made on <strong>ASSUME</strong> pragmas affect optimization of the program.</td>
</tr>
<tr>
<td><strong>check</strong></td>
<td>The compiler generates code to check the correctness of all assertions marked as certain, and emits a runtime message if the assertion is violated; the program continues if <strong>fatal</strong> is not also specified.</td>
</tr>
<tr>
<td><strong>fatal</strong></td>
<td>When used with <strong>check</strong>, the program will terminate when an assertion marked certain is violated.</td>
</tr>
<tr>
<td><strong>retrospective</strong> [d]</td>
<td>The d parameter is an optional tolerance value, and must be a real positive constant less than 1. The default is &quot;.1&quot;. <strong>retrospective</strong> compiles code to count the truth or falsity of all assertions. Those outside the tolerance value d are listed on output at program termination.</td>
</tr>
<tr>
<td><strong>%none</strong></td>
<td>All <strong>ASSUME</strong> pragmas are ignored.</td>
</tr>
</tbody>
</table>

The compiler default is

```bash
-xassume_control=optimize
```

This means that the compiler recognizes **ASSUME** pragmas and they will affect optimization, but no checking is done.

If specified without parameters, `-xassume_control` implies

```bash
-xassume_control=check,fatal
```

In this case the compiler accepts and checks all certain **ASSUME** pragmas, but they do not affect optimization. Assertions that are invalid cause the program to terminate.

### 3.4.112  `-xautopar`

Synonym for `-autopar`.

### 3.4.113  `-xbinopt={prepare|off}`

**(SPARC)** Prepare binary for post-compilation optimization.

The compiled binary file is enabled for later optimizations, transformations, and analysis by `binopt(1)`. This option may be used when building executables or shared objects, and it must be used with an optimization level of `-O1` or higher to be effective.

There will be a modest increase in the size of the binary file when built with this option, on the order of 5%.

If you compile and link in separate steps, `-xbinopt` must appear on both the compile and link steps.
If not all the source code for an application is compiled with `-xbinopt`, the `-xbinopt` flag should still appear on the final link step that builds the program binary, as shown below:

```f95 -0 program -xbinopt=prepare a.o b.o c.f95```

Only code compiled with `-xbinopt` can be optimized by `binopt(1)`. The default is `-xbinopt=off`.

### 3.4.114 `-xcache=\text{c}`

Define cache properties for the optimizer.

\text{c} must be one of the following:

- `generic`
- `native`
- `s1/l1/a1/[t1]`
- `s1/l1/a1/[t1]:s2/l2/a2/[t2]`
- `s1/l1/a1/[t1]:s2/l2/a2/[t2]:s3/l3/a3/[t3]`

The `si/li/ai/ti` 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`
- `ti`: The number of hardware threads sharing the cache at level `i` (optional).

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>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>generic</code></td>
<td>Define the cache properties for good performance on most processors without any major performance degradation. This is the default.</td>
</tr>
<tr>
<td><code>native</code></td>
<td>Define the cache properties for good performance on this host platform.</td>
</tr>
<tr>
<td><code>s1/l1/a1/[t1]</code></td>
<td>Define level 1 cache properties.</td>
</tr>
<tr>
<td><code>s1/l1/a1/[t1]:s2/l2/a2/[t2]</code></td>
<td>Define levels 1 and 2 cache properties.</td>
</tr>
</tbody>
</table>
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.

### 3.4.115 `-xcheck=keyword`

Generate special runtime checks and initializations.

The `keyword` must be one of the following:

<table>
<thead>
<tr>
<th><code>keyword</code></th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stkovf</code></td>
<td>Turn on runtime checking for stack overflow on subprogram entry. If a stack overflow is detected, a <code>SIGSEGV</code> segment fault will be raised. (SPARC only)</td>
</tr>
<tr>
<td><code>no%stkovf</code></td>
<td>Disable runtime checking for stack overflow. (SPARC only)</td>
</tr>
<tr>
<td><code>init_local</code></td>
<td>Perform special initialization of local variables. The compiler initializes local variables to a value that is likely to cause an arithmetic exception if it is used by the program before it is assigned. Memory allocated by the <code>ALLOCATE</code> statement will also be initialized in this manner. Module variables, <code>SAVE</code> variables, and variables in <code>COMMON</code> blocks are not initialized.</td>
</tr>
<tr>
<td><code>no%init_local</code></td>
<td>Disable local variable initialization. This is the default.</td>
</tr>
<tr>
<td><code>%all</code></td>
<td>Turn on all these runtime checking features.</td>
</tr>
<tr>
<td><code>%none</code></td>
<td>Disable all these runtime checking features.</td>
</tr>
</tbody>
</table>

Stack overflows, especially in multithreaded applications with large arrays allocated on the stack, can cause silent data corruption in neighboring thread stacks. Compile all routines with `-xcheck=stkovf` if stack overflow is suspected. But note that compiling with this flag does not guarantee that all stack overflow situations will be detected since they could occur in routines not compiled with this flag.
### 3.4.116 \texttt{--xchip=c}

Specify target processor for the optimizer.

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 \texttt{--xtarget} option; it is provided to allow overriding a \texttt{-xchip} value implied by the a specific \texttt{-xtarget} option.

Some effects of \texttt{-xchip=c} are:

- Instruction scheduling
- The way branches are compiled
- Choice between semantically equivalent alternatives

The following tables list the valid \texttt{-xchip} processor name values:

**TABLE 3-15  Common \texttt{-xchip} SPARC Processor Names**

<table>
<thead>
<tr>
<th>\texttt{-xchip}</th>
<th>Optimize for</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>most SPARC processors. (This is the default.)</td>
</tr>
<tr>
<td>native</td>
<td>this host platform.</td>
</tr>
<tr>
<td>sparc64vi</td>
<td>SPARC64 VI processor</td>
</tr>
<tr>
<td>sparc64vii</td>
<td>SPARC64 VII processor</td>
</tr>
<tr>
<td>ultra</td>
<td>UltraSPARC processor.</td>
</tr>
<tr>
<td>ultra2</td>
<td>UltraSPARC II processor.</td>
</tr>
<tr>
<td>ultra2e</td>
<td>UltraSPARC IIe processor.</td>
</tr>
<tr>
<td>ultra3i</td>
<td>UltraSPARC IIIi processor.</td>
</tr>
<tr>
<td>ultra3</td>
<td>UltraSPARC III processor.</td>
</tr>
<tr>
<td>ultra3cu</td>
<td>UltraSPARC IIIcu processor.</td>
</tr>
<tr>
<td>ultra3i</td>
<td>UltraSPARC IIIi processor</td>
</tr>
<tr>
<td>ultra4</td>
<td>UltraSPARC IV processor</td>
</tr>
<tr>
<td>ultra4plus</td>
<td>UltraSPARC IV+ processor</td>
</tr>
<tr>
<td>ultraT1</td>
<td>UltraSPARC T1 processor</td>
</tr>
<tr>
<td>ultraT2</td>
<td>UltraSPARC T2 processor</td>
</tr>
<tr>
<td>ultraT2plus</td>
<td>UltraSPARC T2+ processor</td>
</tr>
</tbody>
</table>
On x86 platforms: the -xchip values are pentium, pentium_pro, pentium3, pentium4, generic, opteron, core2, penryn, nehalem, amdfam10, and native.

### 3.4.117 -xcode=keyword

(SPARC) Specify code address space on SPARC platforms.

The values for keyword are:

<table>
<thead>
<tr>
<th>keyword</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs32</td>
<td>Generate 32-bit absolute addresses. Code+data+bss size is limited to $2^{32}$ bytes. This is the default on 32-bit platforms.</td>
</tr>
<tr>
<td>abs44</td>
<td>Generate 44-bit absolute addresses. Code+data+bss size is limited to $2^{44}$ bytes. Available only on 64-bit platforms.</td>
</tr>
<tr>
<td>abs64</td>
<td>Generate 64-bit absolute addresses. Available only on 64-bit platforms.</td>
</tr>
<tr>
<td>pic13</td>
<td>Generate position-independent code (small model). Equivalent to -pic. Permits references to at most $2^{11}$ unique external symbols on 32-bit platforms, $2^{10}$ on 64-bit platforms.</td>
</tr>
<tr>
<td>pic32</td>
<td>Generate position-independent code (large model). Equivalent to -PIC. Permits references to at most $2^{30}$ unique external symbols on 32-bit platforms, $2^{29}$ on 64-bit platforms.</td>
</tr>
</tbody>
</table>

The defaults for not specifying -xcode=keyword explicitly are:

- -xcode=abs32 on 32-bit platforms.
- -xcode=abs44 on 64-bit platforms.

#### 3.4.117.1 Position-Independent Code:

Use -xcode=pic13 or -xcode=pic32 when creating dynamic shared libraries to improve runtime performance.

While the code within a dynamic executable is usually tied to a fixed address in memory, position-independent code can be loaded anywhere in the address space of the process.

When you use position-independent code, relocatable references are generated as an indirect reference through a global offset table. Frequently accessed items in a shared object will benefit from compiling with -xcode=pic13 or -xcode=pic32 by not requiring the large number of relocations imposed by code that is not position-independent.

The size of the global offset table is limited to 8Kb.

There are two nominal performance costs with -xcode={pic13|pic32}: 
A routine compiled with either `-xcode=pic13` or `-xcode=pic32` 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 `pic32`, there are two additional instructions per global and static memory reference.

When considering the above costs, remember that the use of `-xcode=pic13` or `-xcode=pic32` can significantly reduce system memory requirements, due to the effect of library code sharing. Every page of code in a shared library compiled `-xcode=pic13` or `-xcode=pic32` 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 `-xcode=pic13` or `-xcode=pic32` is with the `nm` command:

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

A `.o` file containing position-independent code will contain an unresolved external reference to `_GLOBAL_OFFSET_TABLE_` as marked by the letter U.

To determine whether to use `-xcode=pic13` or `-xcode=pic32`, check the size of the Global Offset Table (GOT) by using `elfdump -c` (see the `elfdump(1)` man page for more information) and for the section header, `sh_name: .got`. The `sh_size` value is the size of the GOT. If the GOT is less than 8,192 bytes, specify `-xcode=pic13`, otherwise specify `-xcode=pic32`.

In general, use the following guidelines to determine how you should use `-xcode`:

- If you are building an executable you should not use `-xcode=pic13` or `-xcode=pic32`.
- If you are building an archive library only for linking into executables you should not use `-xcode=pic13` or `-xcode=pic32`.
- If you are building a shared library, start with `-xcode=pic13` and once the GOT size exceed 8,192 bytes, use `-xcode=pic32`.
- If you are building an archive library for linking into shared libraries you should just use `-xcode=pic32`.

Compiling with the `-xcode=pic13` or `pic32` (or `-pic` or `-PIC`) options is recommended when building dynamic libraries. See the Solaris Linker and Libraries Guide.

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

Enable runtime checking of common block inconsistencies.
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 -xcommonchk=yes 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. -xcommonchk without a value is equivalent to -xcommonchk=yes.

### 3.4.119 -xcrossfile[{1|0}]

Enable optimization and inlining across source files.

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 ensure that the new code is properly inlined. See “3.4.45 -inline=[auto][[,]][no%][/,...][no%]/[i]” on page 71.

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

Any .s assembler source files in the compilation do not participate in the crossfile analysis. Also, the -xcrossfile flag is ignored if compiling with -S.

### 3.4.120 -xdebugformat={dwarf|stabs}

Sun Studio compilers are migrating the format of debugger information from the "stabs" format to the "dwarf" format. The default setting for this release is -xdebugformat=dwarf.

If you maintain software which reads debugging information, you now have the option to transition your tools from the stabs format to the dwarf format.
Use this option as a way of accessing the new format for the purpose of porting tools. There is no need to use this option unless you maintain software which reads debugger information, or unless a specific tool tells you that it requires debugger information in one of these formats.

-xdebugformat=stabs generates debugging information using the stabs standard format.

-xdebugformat=dwarf generates debugging information using the dwarf standard format.

If you do not specify -xdebugformat, the compiler assumes -xdebugformat=dwarf. It is an error to specify the option without an argument.

This option affects the format of the data that is recorded with the -g option. Some the format of that information is also controlled with this option. So -xdebugformat has an effect even when -g is not used.

The dbx and Performance Analyzer software understand both stabs and dwarf format so using this option does not have any effect on the functionality of either tool.

This is a transitional interface so expect it to change in incompatible ways from release to release, even in a minor release. The details of any specific fields or values in either stabs or dwarf are also evolving.

Use the dwarfdump(1) command to determine the format of the debugging information in a compiled object or executable file.

3.4.121 –xdepend

Synonym for -depend.

3.4.122 –xF

Allow function–level reordering by the Performance Analyzer.

Allow the reordering of functions (subprograms) in the core image using the compiler, the performance 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. (The f95 –Mpath option also passes a regular file to the linker; see the description of the f95 –Mpath option.)

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. See the Program Performance Analysis Tools manual for further information on the analyzer.
3.4.123  

-xfilebyteorder=\textit{options}

Support file sharing between little-endian and big-endian platforms.

The flag identifies the byte-order and byte-alignment of data on unformatted I/O files. \textit{options} must specify any combination of the following, but at least one specification must be present:

\textit{little}\texttt{max_align}:\textit{spec}
\textit{big}\texttt{max_align}:\textit{spec}
\textit{native}:\textit{spec}

\texttt{max_align} declares the maximum byte alignment for the target platform. Permitted values are 1, 2, 4, 8, and 16. The alignment applies to Fortran VAX structures and Fortran derived types that use platform-dependent alignments for compatibility with C language structures.

\textit{little} specifies a "little-endian" file on platforms where the maximum byte alignment is \textit{max_align}. For example, \textit{little4} specifies a 32-bit x86 file, while \textit{little16} describes a 64-bit x86 file.

\textit{big} specifies a "big-endian" file with a maximum alignment of \textit{max_align}. For example, \textit{big8} describes a 32-bit SPARC file, while \textit{big16} describes a 64-bit SPARC file.

\textit{native} specifies a "native" file with the same byte order and alignment used by the compiling processor platform. The following are assumed to be "native":

<table>
<thead>
<tr>
<th>Platform</th>
<th>&quot;native&quot; corresponds to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit SPARC</td>
<td>\textit{big8}</td>
</tr>
<tr>
<td>64-bit SPARC</td>
<td>\textit{big16}</td>
</tr>
<tr>
<td>32-bit x86</td>
<td>\textit{little4}</td>
</tr>
<tr>
<td>64-bit x86</td>
<td>\textit{little16}</td>
</tr>
</tbody>
</table>

\textit{spec} must be a comma-separated list of the following:

\texttt{all}
\texttt{unit}
\texttt{filename}

\texttt{all} refers to all files and logical units except those opened as \texttt{"SCRATCH"}, or named explicitly elsewhere in the \texttt{-xfilebyteorder} flag. \texttt{all} can only appear once.

\texttt{unit} refers to a specific Fortran unit number opened by the program.
filename refers to a specific Fortran file name opened by the program.

### 3.4.123.1 Examples:

- `-xfilebyteorder=little4:1,2,afilename.in,big8:9,bfilename.out,12`
- `-xfilebyteorder=little8:%all,big16:12`

### 3.4.123.2 Notes:

This option does not apply to files opened with STATUS="SCRATCH". I/O operations done on these files are always with the byte-order and byte-alignment of the native processor.

The first default, when `-xfilebyteorder` does not appear on the command line, is `-xfilebyteorder=native:%all`.

A file name or unit number can be declared only once in this option.

When `-xfilebyteorder` does appear on the command line, it must appear with at least one of the little, big, or native specifications.

Files not explicitly declared by this flag are assumed to be native files. For example, compiling with `-xfilebyteorder=little4:filename` declares `filename` to be a little-endian 32-bit x86 file with a 4-byte maximum data alignment. All other files in the program are native files.

When the byte-order specified for a file is the same as the native processor but a different alignment is specified, the appropriate padding will be used even though no byte swapping is done. For example, this would be the case when compiling with `-m64` for 64-bit x86 platforms and `-xfilebyteorder=little4:filename` is specified.

The declared types in data records shared between big-endian and little-endian platforms must have the same sizes. For example, a file produced by a SPARC executable compiled with `-xtypemap=integer:64,real:64,double:128` cannot be read by an x86 executable compiled with `-xtypemap=integer:64,real:64,double:64` since the default double precision data types will have different sizes. (However, note that starting with the release of Sun Studio 12 Update 1, `double:128` is accepted on x64 processors.)

If an option that changes the alignment of the components within a VAX structure (such as `-vax=struct_align`) or a derived type (such as `-aligncommon` or `-dalign`) is used on one platform, the same alignment option has to be used on other platforms sharing the same unformatted data file whose content is affected by the alignment option.

An I/O operation with an entire UNION/MAP data object on a file specified as non-native will result in a runtime I/O error. You can only execute I/O operations using the individual members of the MAP (and not an entire VAX record containing the UNION/MAP) on non-native files.
3.4.124  

-\texttt{xhasc}\{\texttt{yes|no}\}

Treat Hollerith constant as a character string in an actual argument list.

With \texttt{-xhasc=yes}, the compiler treats Hollerith constants as character strings when they appear as an actual argument on a subroutine or function call. This is the default, and complies with the Fortran standard. (The actual call list generated by the compiler contains hidden string lengths for each character string.)

With \texttt{-xhasc=no}, Hollerith constants are treated as typeless values in subprogram calls, and only their addresses are put on the actual argument list. (No string length is generated on the actual call list passed to the subprogram.)

Compile routines with \texttt{-xhasc=no} if they call a subprogram with a Hollerith constant and the called subprogram expects that argument as \texttt{INTEGER} (or anything other than \texttt{CHARACTER}).

Example:

```fortran
demo% cat hasc.f
   call z(4habcd, 'abcdefg')
   end
   subroutine z(i, s)
     integer i
     character *(*) s
     print *, "string length = ", len(s)
     return
   end

demo% f95 -o has0 hasc.f

demo% has0
   string length = 4  \textit{\textless\textgreater should be 7}

demo% f95 -o has1 -xhasc=no hasc.f

demo% has1
   string length = 7  \textit{\textless\textgreater now correct length for s}
```

Passing \texttt{4habcd} to \texttt{z} is handled correctly by compiling with \texttt{-xhasc=no}.

This flag is provided to aid porting legacy Fortran 77 programs.

3.4.125  

-\texttt{xhelp}\{\texttt{readme|flags}\}

Show summary help information.

\texttt{-xhelp=readme} Show the online \texttt{README} file for this release of the compiler.

\texttt{-xhelp=flags} List the compiler option flags. Equivalent to \texttt{-help}. 

### 3.4.126 \(-xhwcprof[=\{enable \mid disable\}]\)

**(SPARC)** Enable compiler support for dataspace profiling.

With `-xhwcprof` enabled, the compiler generates information that helps tools associate profiled load and store instructions with the data types and structure members (in conjunction with symbolic information produced with `-g`) to which they refer. It associates profile data with the data space of the target, rather than the instruction space, and provides insight into behavior that is not easily obtained from only instruction profiling.

While you can compile a specified set of object files with `-xhwcprof`, this option is most useful when applied to all object files in the application. This will provide coverage to identify and correlate all memory references distributed in the application's object files.

If you are compiling and linking in separate steps, use `-xhwcprof` at link time as well.

An instance of `-xhwcprof=enable` or `-xhwcprof=disable` overrides all previous instances of `-xhwcprof` in the same command line.

- `xhwcprof` is disabled by default. Specifying `-xhwcprof` without any arguments is the equivalent to `-xhwcprof=enable`.

- `-xhwcprof` requires that optimization be turned on and that the debug data format be set to dwarf (`-xdebugformat=dwarf`), which is the default with this release of Sun Studio.

The combination of `-xhwcprof` and `-g` increases compiler temporary file storage requirements by more than the sum of the increases due to `-xhwcprof` and `-g` specified alone.

The following command compiles `example.f` and specifies support for hardware counter profiling and symbolic analysis of data types and structure members using DWARF symbols:

```bash
f95 -c -O -xhwcprof -g example.f
```

For more information on hardware counter-based profiling, see the Sun Studio Performance Analyzer manual.

### 3.4.127 \(-xia[=\{widestneed\mid strict\}]\)

**(Solaris)** Enable interval arithmetic extensions and set a suitable floating-point environment.

The default if not specified is `-xia=widestneed`.

Fortran extensions for interval arithmetic calculations are detailed in the *Interval Arithmetic Programming Reference*. See also "3.4.130 \(-xinterval[=\{widestneed\mid strict\mid no\}]\)" on page 112.
The `-xia` flag is a macro that expands as follows:

<table>
<thead>
<tr>
<th><code>-xia</code></th>
<th><code>-xinterval=widestneed -ftrap=%none -fns=no -fsimple=0</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-xia=strict</code></td>
<td><code>-xinterval=strict -ftrap=%none -fns=no -fsimple=0</code></td>
</tr>
</tbody>
</table>

**3.4.128** `-xinline= list`

Synonym for `-inline`.

**3.4.129** `-xinstrument=[%no]datarace`

Specify this option to compile and instrument your program for analysis by the Thread Analyzer.

(For information on the Thread Analyzer, see `tha(1)` for details.)

By compiling with this option you can then use the Performance Analyzer to run the instrumented program with `collect -r races` to create a data-race-detection experiment.

You can run the instrumented code standalone but it runs more slowly.

Specify `-xinstrument=no%datarace` to turn off this feature. This is the default.

`-xinstrument` must be specified with an argument.

If you compile and link in separate steps, you must specify `-xinstrument=datarace` in both the compilation and linking steps.

This option defines the preprocessor token `__THA_NOTIFY`. You can specify `#ifdef __THA_NOTIFY` to guard calls to `libtha(3)` routines.

This option also sets `-g`.

**3.4.130** `-xinterval[={widestneed|strict|no}]`

(Solaris) Enable interval arithmetic extensions.

The optional value can be one of either `no`, `widestneed` or `strict`. The default if not specified is `widestneed`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>no</code></td>
<td>Interval arithmetic extensions not enabled.</td>
</tr>
</tbody>
</table>
### 3.4.131 \(-xipo=\{0|1|2\}\)

Perform interprocedural optimizations.

Performs whole-program optimizations by invoking an interprocedural analysis pass. Unlike \(-xcrossfile\), \(-xipo\) will perform optimizations across all object files in the link step, and is not limited to just the source files on the compile command.

\(-xipo\) is particularly useful when compiling and linking large multi-file applications. Object files compiled with this flag have analysis information compiled within them that enables interprocedural analysis across source and pre-compiled program files. However, analysis and optimization is limited to the object files compiled with \(-xipo\), and does not extend to object files on libraries.

\(-xipo=0\) disables, and \(-xipo=1\) enables, interprocedural analysis. \(-xipo=2\) adds interprocedural aliasing analysis and memory allocation and layout optimizations to improve cache performance. The default is \(-xipo=0\), and if \(-xipo\) is specified without a value, \(-xipo=1\) is used.

When compiling with \(-xipo=2\), there should be no calls from functions or subroutines compiled without \(-xipo=2\) (for example, from libraries) to functions or subroutines compiled with \(-xipo=2\).

As an example, if you interpose on the function `malloc()` and compile your own version of `malloc()` with \(-xipo=2\), all the functions that reference `malloc()` in any library linked with your code would also have to be compiled with \(-xipo=2\). Since this might not be possible for system libraries, your version of `malloc` should not be compiled with \(-xipo=2\).

When compiling and linking are performed in separate steps, \(-xipo\) must be specified in both steps to be effective.

Example using \(-xipo\) in a single compile/link step:

```
demo% f95 -xipo -xO4 -o prog part1.f part2.f part3.f
```

The optimizer performs crossfile inlining across all three source files. This is done in the final link step, so the compilation of the source files need not all take place in a single compilation and could be over a number of separate compilations, each specifying \(-xipo\).
Example using \texttt{-xipo} in separate compile/link steps:

\begin{verbatim}
demo% f95 -xipo -xO4 -c part1.f part2.f
demo% f95 -xipo -xO4 -c part3.f
demo% f95 -xipo -xO4 -o prog part1.o part2.o part3.o
\end{verbatim}

The object files created in the compile steps have additional analysis information compiled within them to permit crossfile optimizations to take place at the link step.

A restriction is that libraries, even if compiled with \texttt{-xipo} do not participate in crossfile interprocedural analysis, as shown in this example:

\begin{verbatim}
demo% f95 -xipo -xO4 one.f two.f three.f
demo% ar -r mylib.a one.o two.o three.o ...
demo% f95 -xipo -xO4 -o myprog main.f four.f mylib.a
\end{verbatim}

Here interprocedural optimizations will be performed between \texttt{one.f}, \texttt{two.f} and \texttt{three.f}, and between \texttt{main.f} and \texttt{four.f}, but not between \texttt{main.f} or \texttt{four.f} and the routines on \texttt{mylib.a}. (The first compilation may generate warnings about undefined symbols, but the interprocedural optimizations will be performed because it is a compile and link step.)

Other important information about \texttt{-xipo}:

- requires at least optimization level \texttt{-xO4}
- conflicts with \texttt{-xcrossfile}; if used together will result in a compilation error
- Building executables compiled with \texttt{-xipo} using a parallel make tool can cause problems if object files used in the build are common to the link steps running in parallel. Each link step should have its own copy of the object file being optimized prior to linking.
- objects compiled without \texttt{-xipo} can be linked freely with objects compiled with \texttt{-xipo}.
- The \texttt{-xipo} option generates significantly larger object files due to the additional information needed to perform optimizations across files. However, this additional information does not become part of the final executable binary file. Any increase in the size of the executable program will be due to the additional optimizations performed.
- In this release, crossfile subprogram inlining is the only interprocedural optimization performed by \texttt{-xipo}.
- \texttt{.s} assembly language files do not participate in interprocedural analysis.
- The \texttt{-xipo} flag is ignored if compiling with \texttt{-S}. 
When Not To Compile With -xipo:

Working with the set of object files in the link step, the compiler tries to perform whole-program analysis and optimizations. For any function or subroutine foo() defined in this set of object files, the compiler makes the following two assumptions:

(1) At runtime, foo() will not be called explicitly by another routine defined outside this set of object files, and

(2) calls to foo() from any routine in the set of object files will be not be interposed upon by a different version of foo() defined outside this set of object files.

If assumption (1) is not true for the given application, do not compile with -xipo=2. If assumption (2) is not true, do not compile with either -xipo=1 or -xipo=2.

As an example, consider interposing on the function malloc() with your own source version and compiling with -xipo=2. Then all the functions in any library that reference malloc() that are linked with your code would have to also be compiled with -xipo=2 and their object files would need to participate in the link step. Since this might not be possible for system libraries, your version of malloc should not be compiled with -xipo=2.

As another example, suppose that you build a shared library with two external calls, foo() and bar() inside two different source files, and bar() calls foo() inside its body. If there is a possibility that the function call foo() could be interposed at runtime, then compile neither source file for foo() or bar() with -xipo=1 or -xipo=2. Otherwise, foo() could be inlined into bar(), which could cause incorrect results when compiled with -xipo.

3.4.132 -xipo_archive[=none|readonly|writeback]]

(SPARC) Allow crossfile optimization to include archive (.a) libraries.

The value must be one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>No processing of archive files is performed. The compiler does not apply cross-module inlining or other cross-module optimizations to object files compiled using -xipo and extracted from an archive library at link time. To do that, both -xipo and either -xipo_archive=readonly or -xipo_archive=writeback must be specified at link time.</td>
</tr>
</tbody>
</table>
### 3.4.133 —xjobs=n

Compile with multiple processors.

Specify the -xjobs option to set how many processes the compiler creates to complete its work. This option can reduce the build time on a multi-cpu machine. In this release of the f95 compiler, -xjobs works only with the -xipo option. When you specify -xjobs=n, the interprocedural optimizer uses n as the maximum number of code generator instances it can invoke to compile different files.

Generally, a safe value for n is 1.5 multiplied by the number of available virtual processors. Using a value that is many times the number of available virtual processors can degrade performance because of context switching overheads among spawned jobs. Also, using a very high number can exhaust the limits of system resources such as swap space.

You must always specify -xjobs with a value. Otherwise an error diagnostic is issued and compilation aborts.

Multiple instances of -xjobs on the command line override each other until the rightmost instance is reached.

The following example compiles more quickly on a system with two processors than the same command without the -xjobs option.

```
example% f95 -xipo -xO4 -xjobs=3 t1.f t2.f t3.f
```
### 3.4.134  `-xknown_lib=library_list`

Recognize calls to a known library.

When specified, the compiler treats references to certain known libraries as intrinsics, ignoring any user-supplied versions. This enables the compiler to perform optimizations over calls to library routines based on its special knowledge of that library.

The `library_list` is a comma-delimited list of keywords currently to `blas`, `blas1`, `blas2`, `blas3`, and `intrinsics`. The compiler recognizes calls to the following BLAS1, BLAS2, and BLAS3 library routines and is free to optimize appropriately for the Sun Performance Library implementation. The compiler will ignore user-supplied versions of these library routines and link to the BLAS routines in the Sun Performance Library.

<table>
<thead>
<tr>
<th><code>-xknown_lib=</code></th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>blas1</td>
<td>The compiler recognizes calls to the following BLAS1 library routines:</td>
</tr>
<tr>
<td></td>
<td>caxpy, ccopy, cdotc, cdotu, crotg, cscal, csrot, csscal, cswap, dasum, daxpy, dcopy, ddot, drot, drotm, drotmg, dscal, dswap, dnrm2, dzasum, dznm2, icamax, idamax, isamax, nasmx, nasum, saxpy, scasum, ssctrn2, scopy, sdot, ssdmt, ssnrm2, ssrot, ssrotm, strot, strotmg, stsscal, stswap, zsasum, zscal, zswap, zswap</td>
</tr>
<tr>
<td>blas2</td>
<td>The compiler recognizes calls to the following BLAS2 library routines:</td>
</tr>
<tr>
<td></td>
<td>cgemv, cdgerc, cdgeru, ctrmv, ctrsv, dgemv, dger, dsymv, dsyr, dsyr2, dsyr2k, dsyr2k, dtrmv, dtrsv, sgemv, sgemv, sger, ssymv, ssyr, ssyr2, ssyr2k, stmv, stsv, ststr, stsv, zgemv, zgerc, zgeru, ztrmv, ztrsv</td>
</tr>
<tr>
<td>blas3</td>
<td>The compiler recognizes calls to the following BLAS2 library routines:</td>
</tr>
<tr>
<td></td>
<td>cgemm, cdgemm, csyr2k, csyrk, ctrmm, dgemm, dsyrmv, dsyr2k, dsyrk, dtrmm, dtrmm, s gemm, ssymv, ssyr2k, ssyrk, strmm, stmm, zgemm, zsymm, zsyrm, zsymm, zsyrk, ztrmm, ztrmm</td>
</tr>
<tr>
<td>blas</td>
<td>Selects all the BLAS routines. Equivalent to <code>-xknown_lib=blas1,blas2,blas3</code></td>
</tr>
<tr>
<td>intrinsics</td>
<td>The compiler ignores any explicit <code>EXTERNAL</code> declarations for Fortran intrinsics, thereby ignoring any user-supplied intrinsic routines. (See the Fortran Library Reference for lists of intrinsic function names.)</td>
</tr>
</tbody>
</table>

### 3.4.135  `-xlang=f77`

(SPARC) Prepare for linking with runtime libraries compiled with earlier versions of `f77`.

`f95 -xlang=f77` implies linking with the `f77compat` library, and is a shorthand way for linking `f95` object files with older Fortran 77 object files. Compiling with this flag insures the proper runtime environment.
Use `f95 -xlang=f77` when linking `f95` and `f77` compiled objects together into a single executable.

Note the following when compiling with `-xlang`:

- Do not compile with both `-xnolib` and `-xlang`.
- When mixing Fortran object files with C++, link using the C++ compiler and specify `-xlang=f95` on the `CC` command line.
- When mixing C++ objects with Fortran object files compiled with any of the parallelization options, the linking `CC` command line must also specify `-mt`.

### 3.4.136 `-xlibmil`

Synonym for `-libmil`.

### 3.4.137 `-xlibmopt`

Use library of optimized math routines.

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.

### 3.4.138 `-xic_lib=sunperf`

Link with the Sun Performance Library.

For example:

```
f95 -o pgx -fast pgx.f -xic_lib=sunperf
```

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

This option must be used to link with the Sun Performance Library. (See the *Sun Performance Library User’s Guide*.)

### 3.4.139 `-xicinfo`

*(Obsolete) Silently ignored by compiler.*
3.4.140 \texttt{-xlinkopt[=\{1\mid 2\mid 0\}]} \\

(\textsc{sparc}) Perform link-time optimizations on relocatable object files.

The post-optimizer performs a number of advanced performance optimizations on the binary object code at link-time. The optional value sets the level of optimizations performed, and must be \(0\), \(1\), or \(2\).

\begin{tabular}{|c|p{15cm}|}
\hline
0 & The post-optimizer is disabled. (This is the default.) \\
\hline
1 & Perform optimizations based on control flow analysis, including instruction cache coloring and branch optimizations, at link time. \\
\hline
2 & Perform additional data flow analysis, including dead-code elimination and address computation simplification, at link time. \\
\hline
\end{tabular}

Specifying the \texttt{-xlinkopt} flag without a value implies \texttt{-xlinkopt=1}.

These optimizations are performed at link time by analyzing the object binary code. The object files are not rewritten but the resulting executable code may differ from the original object codes.

This option is most effective when used to compile the whole program, and with profile feedback.

When compiling in separate steps, \texttt{-xlinkopt} must appear on both compile and link steps.

\begin{verbatim}
demo% f95 -c -xlinkopt a.f95 b.f95
demo% f95 -o myprog -xlinkopt=2 a.o b.o
\end{verbatim}

Note that the level parameter is only used when the compiler is linking. In the example above, the postoptimization level used is \(2\) even though the object binaries were compiled with an implied level of \(1\).

The link-time post-optimizer cannot be used with the incremental linker, \texttt{ild}. The \texttt{-xlinkopt} flag will set the default linker to be \texttt{ld}. Enabling the incremental linker explicitly with the \texttt{-xildon} flag will disable the \texttt{-xlinkopt} option if both are specified together.

For the \texttt{-xlinkopt} option to be useful, at least some, but not necessarily all, of the routines in the program must be compiled with this option. The optimizer can still perform some limited optimizations on object binaries not compiled with \texttt{-xlinkopt}.

The \texttt{-xlinkopt} option will optimize code coming from static libraries that appear on the compiler command line, but it will skip and not optimize code coming from shared (dynamic) libraries that appear on the command line. You can also use \texttt{-xlinkopt} when building shared libraries (compiling with \texttt{-G}).
The link-time post-optimizer is most effective when used with run-time profile feedback. Profiling reveals the most and least used parts of the code and directs the optimizer to focus its effort accordingly. This is particularly important with large applications where optimal placement of code performed at link time can reduce instruction cache misses. Typically, this would be compiled as shown below:

```
demo% f95 -o progt -xO5 -xprofile=collect:prog file.f95
demo% progt
demo% f95 -o prog -xO5 -xprofile=use:prog -xlinkopt file.95
```

For details on using profile feedback, see the `-xprofile` option.

Note that compiling with this option will increase link time slightly. Object file sizes also increase, but the size of the executable remains the same. Compiling with the `-xlinkopt` and `-g` flags increases the size of the executable by including debugging information.

### 3.4.141 `–xloopinfo`

Synonym for `-loopinfo`.

### 3.4.142 `–xmaxopt[=n]`

Enable optimization pragma and set maximum optimization level.

`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. See “2.3.1.5 The OPT Directive” on page 33.

If this pragma appears with an optimization level greater than the maximum level on the `-xmaxopt` flag, the compiler uses the level set by `-xmaxopt`.

### 3.4.143 `–xmemalign[=<a><b>]]`

(SPARC) Specify maximum assumed memory alignment and behavior of misaligned data accesses.

For memory accesses where the alignment is determinable at compile time, the compiler will generate the appropriate load/store instruction sequence for that data alignment.

For memory accesses where the alignment cannot be determined at compile time, the compiler must assume an alignment to generate the needed load/store sequence.
The `-xmemalign` flag allows the user to specify the maximum memory alignment of data to be assumed by the compiler for those indeterminate situations. It also specifies the error behavior at runtime when a misaligned memory access does take place.

The value specified consists of two parts: a numeric alignment value, `<a>`, and an alphabetic behavior flag, `<b>`.

Allowed values for alignment, `<a>`, are:

1. Assume at most 1-byte alignment.
2. Assume at most 2-byte alignment.
4. Assume at most 4-byte alignment.
8. Assume at most 8-byte alignment.
16. Assume at most 16-byte alignment.

Allowed values for error behavior on accessing misaligned data, `<b>`, are:

i. Interpret access and continue execution
s. Raise signal SIGBUS
f. On 64–bit platforms, raise signal SIGBUS only for alignments less or equal to 4, otherwise interpret access and continue execution. On other platforms f is equivalent to i.

The defaults when compiling without `-xmemalign` specified are:

- 8i for 32–bit platforms
- 8s for 64–bit platforms with C and C++
- 8f for 64–bit platforms with Fortran

The default for `-xmemalign` appearing without a value is 1i for all platforms.

Note that `-xmemalign` itself does not force any particular data alignment to take place. Use `-dalign` or `-aligncommon` to force data alignment.

You must also specify `-xmemalign` whenever you link to an object file that was compiled with a `b` value of either i or f.

The `-dalign` option is a macro:

- `-dalign` is a macro for: `-xmemalign=8s -aligncommon=16`

Do not use `-aligncommon=1` with `-xmemalign` as these declarations will conflict and could cause a segmentation fault on some platforms and configurations.

See “3.4.2 –aligncommon=[1|2|4|8|16]” on page 51 for details.
3.4.144 –xmodel=[small|kernel|medium]

(x86) Specify the data address model for shared objects on Solaris x64 platforms.

The –xmodel option enables the compiler to create 64-bit shared objects for the Solaris x64 platforms and should only be specified for the compilation of such objects.

This option is valid only when –m64 is specified on 64-bit enabled x86 platforms (“x64”).

small This option generates code for the small model in which the virtual address of code executed is known at link time and all symbols are known to be located in the virtual addresses in the range from 0 to $2^{31} - 2^{24} - 1$.

kernel Generates code for the kernel model in which all symbols are defined to be in the range from $2^{64} - 2^{31}$ to $2^{64} - 2^{24}$.

medium Generates code for the medium model in which no assumptions are made about the range of symbolic references to data sections. Size and address of the text section have the same limits as the small code model. Applications with large amounts of static data might require –xmodel=medium when compiling with –m64.

If you do not specify –xmodel, the compiler assumes –xmodel=small. Specifying –xmodel without an argument is an error.

It is not necessary to compile all routines with this option as long as you ensure that objects being accessed are within range.

3.4.145 –xnolib

Synonym for –nolib.

3.4.146 –xnolibmil

Synonym for –nolibmil.

3.4.147 –xnolibmopt

Do not use fast math library.

Use with –fast to override linking the optimized math library:

f95 -fast -xnolibmopt …
3.4.148 \( -xO n \)

Synonym for \(-on\).

3.4.149 \(-xopenmp\)\([\{parallel|noopt|none\}]\)

Enable explicit parallelization with Fortran OpenMP Version 3.0 directives.

The flag accepts the following optional keyword suboptions:

- **parallel**
  - Enables recognition of OpenMP pragmas, and the program is parallelized accordingly.
  - The minimum optimization level for \(-xopenmp=parallel\) is \(-xO3\). The compiler changes the optimization from a lower level to \(-xO3\) if necessary, and issues a warning.
  - Defines preprocessor token \_OPENMP
  - Invokes \(-stackvar\) automatically.

- **noopt**
  - Enables recognition of OpenMP pragmas, and the program is parallelized accordingly.
  - The compiler does not raise the optimization level if it is lower than \(-xO3\). If you explicitly set the optimization to a level lower than \(-xO3\), as in \(-xO2 -xopenmp=noopt\) the compiler will issue an error. If you do not specify an optimization level with \(-xopenmp=noopt\), the OpenMP pragmas are recognized, the program is parallelized accordingly, but no optimization is done.
  - Defines preprocessor token \_OPENMP
  - Invokes \(-stackvar\) automatically.

- **none**
  - Disables recognition of OpenMP pragmas and does not change the optimization level. (This is the compiler’s default.)

\(-xopenmp\) specified without a suboption keyword is equivalent to \(-xopenmp=parallel\). Note that this default might change in later releases.

To debug OpenMP programs with dbx, compile with \(-g -xopenmp=noopt\) to be able to breakpoint within parallel regions and display the contents of variables.

The OpenMP directives are summarized in the OpenMP API User’s Guide.
To run a parallelized program in a multithreaded environment, you must set the `PARALLEL` (or `OMP_NUM_THREADS`) environment variable prior to execution. This tells the runtime system the maximum number of threads the program can create. The default is 1. In general, set the `PARALLEL` or `OMP_NUM_THREADS` variable to the number of available virtual processors on the target platform.

To enable nested parallelism, you must set the `OMP_NESTED` environment variable to `TRUE`. Nested parallelism is disabled by default. See the Sun Studio OpenMP API User’s Guide for details on nested parallelism.

OpenMP requires the definition of the preprocessor symbol `_OPENMP` to have the decimal value YYYYMM where YYYY and MM are the year and month designations of the version of the OpenMP Fortran API that the implementation supports. For the current Sun Studio release the value is 200830 for OpenMP version 3.0.

When compiling and linking in separate steps, also specify `-xopenmp` on the link step. This is especially important when compiling libraries that contain OpenMP directives.

### 3.4.150 -xpad

Synonym for `-pad`.

### 3.4.151 -xpagesize=size

Set the preferred page size for the stack and the heap.

On SPARC platforms, the `size` value must be one of the following:

- 8K
- 64K
- 512K
- 4M
- 32M
- 256M
- 2G
- 16G
- or default

On x86 platforms, the `size` value must be one of the following:

- 4K
- 2M
- 4M
- or default

For example: `-xpagesize=4M`

Not all these page sizes are supported on all platforms and depend on the architecture and Solaris environment. The page size specified must be a valid page size for the Solaris operating environment on the target platform, as returned by `getpagesize(3C)`. If it is not, the request will be silently ignored at run-time. The Solaris environment offers no guarantee that the page size request will be honored.

You can use `pmap(1)` or `meminfo(2)` to determine if your running program received the requested page size.
If you specify `-xpagesize=default`, the flag is ignored; `-xpagesize` specified without a `size` value is equivalent to `-xpagesize=default`.

This option is a macro for the combination `-xpagesize_heap=size -xpagesize_stack=size`. These two options accept the same arguments as `-xpagesize`. You can set them both with the same value by specifying `-xpagesize=size` or you can specify them individually with different values.

Compiling with this flag has the same effect as setting the `LD_PRELOAD` environment variable to `mpss.so.1` with the equivalent options, or running the Solaris 9 command `ppgsz(1)` with the equivalent options, before starting the program. See the Solaris 9 man pages for details.

### 3.4.152 `-xpagesize_heap=size`

Set the preferred page size for the heap. The `size` value is the same as described for `-xpagesize`. See `-xpagesize` for details.

### 3.4.153 `-xpagesize_stack=size`

(SPARC) Set the preferred page size for the stack. The `size` value is the same as described for `-xpagesize`. See `-xpagesize` for details.

### 3.4.154 `-xpec[={yes|no}]`

Generate a PEC (Portable Executable Code) binary.

PEC binaries may be used with Automatic Tuning System, ATS. More information about ATS is available at [http://cooltools.sunsource.net/ats/index.html](http://cooltools.sunsource.net/ats/index.html).

A binary built with `-xpec` is usually 5 to 10 times larger than if it is built without. The default is `-xpec=no`. Without an argument, `-xpec` is equivalent to `-xpec=yes.`
3.4.155 \(-xpg\)
Synonym for \(-pg\).

3.4.156 \(-xpp=\{fpp|cpp\}\)
Select source file preprocessor.
The default is \(-xpp=fpp\).
The compilers use \(fpp\) to preprocess \(.F\), \(.F95\), or \(.F03\) source files. This preprocessor is appropriate for Fortran. Previous versions used the standard C preprocessor \(cpp\). To select \(cpp\), specify \(-xpp=cpp\).

3.4.157 \(-xprefetch[=a[,a]]\)
Enable prefetch instructions on those architectures that support prefetch.
See “2.3.1.7 The \texttt{PREFETCH} Directives” on page 34 for a description of the Fortran \texttt{PREFETCH} directives.

\(a\) must be one of the following:
\begin{itemize}
  \item \texttt{auto} Enable automatic generation of prefetch instructions
  \item \texttt{no\textbackslash auto} Disable automatic generation of prefetch instructions
  \item \texttt{explicit} Enable explicit prefetch macros (SPARC only)
  \item \texttt{no\textbackslash explicit} Disable explicit prefetch macros (SPARC only)
  \item \texttt{latx\textcolon factor} (SPARC) Adjust the compiler’s assumed prefetch-to-load and prefetch-to-store latencies by the specified factor. The factor must be a positive floating-point or integer number.
\end{itemize}

If you are running computationally intensive codes on large SPARC multiprocessors, you might find it advantageous to use \(-xprefetch=latx\textcolon factor\). This option instructs the code generator to adjust the default latency time between a prefetch and its associated load or store by the specified factor.

The prefetch latency is the hardware delay between the execution of a prefetch instruction and the time the data being prefetched is available in the
cache. The compiler assumes a prefetch latency value when determining how far apart to place a prefetch instruction and the load or store instruction that uses the prefetched data.

**Note** – The assumed latency between a prefetch and a load may not be the same as the assumed latency between a prefetch and a store.

The compiler tunes the prefetch mechanism for optimal performance across a wide range of machines and applications. This tuning may not always be optimal. For memory-intensive applications, especially applications intended to run on large multiprocessors, you may be able to obtain better performance by increasing the prefetch latency values. To increase the values, use a factor that is greater than 1. A value between .5 and 2.0 will most likely provide the maximum performance.

For applications with datasets that reside entirely within the external cache, you may be able to obtain better performance by decreasing the prefetch latency values. To decrease the values, use a factor that is less than 1.

To use the `-xprefetch=latx:factor` option, start with a factor value near 1.0 and run performance tests against the application. Then increase or decrease the factor, as appropriate, and run the performance tests again. Continue adjusting the factor and running the performance tests until you achieve optimum performance. When you increase or decrease the factor in small steps, you will see no performance difference for a few steps, then a sudden difference, then it will level off again.

<table>
<thead>
<tr>
<th>yes</th>
<th><code>-xprefetch=yes</code> is the same as <code>-xprefetch=auto,explicit</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td><code>-xprefetch=no</code> is the same as <code>-xprefetch=no%auto,no%explicit</code></td>
</tr>
</tbody>
</table>

With `-xprefetch`, `-xprefetch=auto`, and `-xprefetch=yes`, the compiler is free to insert prefetch instructions into the code it generates. This may result in a performance improvement on architectures that support prefetch.

### 3.4.157.1 Defaults:

If `-xprefetch` is not specified, `-xprefetch=no%auto,explicit` is assumed.

If only `-xprefetch` is specified, `-xprefetch=auto,explicit` is assumed.

The default of `no%auto` is assumed unless explicitly overridden with the use of `-xprefetch` without any arguments or with an argument of `auto` or `yes`. For example, `-xprefetch=explicit` is the same as `-xprefetch=explicit,no%auto`. 
The default of `explicit` is assumed unless explicitly overridden with an argument of `no%explicit` or an argument of `no`. For example, `-xprefetch=auto` is the same as `-xprefetch=auto,explicit`.

If automatic prefetching is enabled, such as with `-xprefetch` or `-xprefetch=yes`, but a latency factor is not specified, then `-xprefetch=latx:1.0` is assumed.

### 3.4.157.2 Interactions:

With `-xprefetch=explicit`, the compiler will recognize the directives:

```
$PRAGMA SUN_PREFETCH_READ_ONCE (name)
$PRAGMA SUN_PREFETCH_READ_MANY (name)
$PRAGMA SUN_PREFETCH_WRITE_ONCE (name)
$PRAGMA SUN_PREFETCH_WRITE_MANY (name)
```

The `-xchip` setting effects the determination of the assumed latencies and therefore the result of a `latx:factor` setting.

The `latx:factor` suboption is valid only when automatic prefetching (`auto`) is enabled on SPARC processors.

### 3.4.157.3 Warnings:

Explicit prefetching should only be used under special circumstances that are supported by measurements.

Because the compiler tunes the prefetch mechanism for optimal performance across a wide range of machines and applications, you should only use `-xprefetch=latx:factor` when the performance tests indicate there is a clear benefit. The assumed prefetch latencies may change from release to release. Therefore, retesting the effect of the latency factor on performance whenever switching to a different release is highly recommended.

### 3.4.158 `-xprefetch_auto_type=indirect_array_access`

Generate indirect prefetches for a data arrays accessed indirectly.

Generates indirect prefetches for the loops indicated by the option `-xprefetch_level={1|2|3}` in the same fashion the prefetches for direct memory accesses are generated. The prefix `no%` can be added to negate the declaration.

The default is `-xprefetch_auto_type=no%indirect_array_access`.

Requires `-xprefetch=auto` and an optimization level `-xO3` or higher.
Options such as `-xdepend` can affect the aggressiveness of computing the indirect prefetch candidates and therefore the aggressiveness of the automatic indirect prefetch insertion due to better memory alias disambiguation information.

### 3.4.159 `-xprefetch_level={1|2|3}`

Control the automatic generation of prefetch instructions.

This option is only effective when compiling with:

- `-xprefetch=auto`,
- with optimization level 3 or greater,
- on a platform that supports prefetch.

The default for `-xprefetch=auto` without specifying `-xprefetch_level` is level 2.

Prefetch level 2 generates additional opportunities for prefetch instructions than level 1. Prefetch level 3 generates additional prefetch instructions than level 2.

Prefetch levels 2 and 3 may not be effective on older SPARC or x86 platforms.

### 3.4.160 `–xprofile={collect[:name]|use[:name]|tcov}`

Collect or optimize with runtime profiling data, or perform basic block coverage analysis.

Compiling with high optimization levels (`-xO5`) is enhanced by providing the compiler with runtime performance feedback. To produce the profile feedback the compiler needs to do its best optimizations, you must compile first with `-xprofile=collect`, run the executable against a typical data set, and then recompile at the highest optimization level and with `-xprofile=use`.

See also the `f95(1)` man page for additional information about `–xprofile` and environment variables.

- `collect[:name]` 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 `name` is the optional name of the directory in which profile data will be stored when the program is executed. If specified, `name` should be an absolute UNIX pathname. If `name` is not specified, profile data for a profiled program named `program` will be stored in a directory named `program.profile` in the current working directory at the time that `program` is executed.
At runtime a program compiled with `-xprofile=collect:name` will create by default the subdirectory `name.profile` to hold the runtime feedback information. The program writes its runtime profile data to the file named `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.

You can set the environment variables `SUN_PROFDATA` and `SUN_PROFDATA_DIR` to control the file and directory where a program compiled with `-xprofile=collect` writes its runtime profile data. With these variables set, the program compiled with `-xprofile=collect` writes its profile data to `$SUN_PROFDATA_DIR/$SUN_PROFDATA`.

These environment variables similarly control the path and names of the profile data files written by `tcov`, as described in the `tcov(1)` man page.

Profile collection is "MT-safe". That is, profiling a program that does its own multitasking by compiling with `-mt` and calling the multitasking library directly will give accurate results.

When compiling and linking in separate steps, the link step must also specify `-xprofile=collect` if it appears on the compile step.

```
use[:nm]
```

Use execution frequency data to optimize strategically at optimization level `-x05`.

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 profile data 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`.

See also `-xprofile_ircache` for speeding up compilation between the collect and use phases.

See also `-xprofile_pathmap` for controlling where the compiler looks for profile data files.

```
tcov
```

Basic block coverage analysis using "new" style `tcov`. Optimization level must be `-O2` or greater.
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/tcova 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 Program Performance Analysis Tools manual for more details.

Note—The report produced by tcov can be unreliable if there is inlining of subprograms due to -O4 or -inline. Coverage of calls to routines that have been inlined is not recorded.

3.4.161 -xprofile_ircache[=path]

(SPARC) Save and reuse compilation data between collect and use profile phases.

Use with -xprofile=collect|use to improve compilation time during the use phase by reusing compilation data saved from the collect phase.

If specified, path will override the location where the cached files are saved. By default, these files will be saved in the same directory as the object file. Specifying a path is useful when the collect and use phases happen in two different places.

A typical sequence of commands might be:

```
demo% f95 -xO5 -xprofile=collect -xprofile_ircache t1.c t2.c

demo% a.out collects feedback data

demo% f95 -xO5 -xprofile=use -xprofile_ircache t1.c t2.c
```

With large programs, compilation time in the use phase can improve significantly by saving the intermediate data in this manner. But this will be at the expense of disk space, which could increase considerably.
3.4.162 \texttt{-xprofile\_pathmap=collect\_prefix:use\_prefix}

(SPARC) Set path mapping for profile data files.

Use the \texttt{-xprofile\_pathmap} option with the \texttt{-xprofile=use} option.

Use \texttt{-xprofile\_pathmap} when the compiler is unable to find profile data for an object file that is compiled with \texttt{-xprofile=use}, and:

- You are compiling with \texttt{-xprofile=use} into a directory that is not the directory used when previously compiling with \texttt{-xprofile=collect}.
- Your object files share a common basename in the profile but are distinguished from each other by their location in different directories.

The \texttt{collect\_prefix} is the prefix of the UNIX pathname of a directory tree in which object files were compiled using \texttt{-xprofile=collect}.

The \texttt{use\_prefix} is the prefix of the UNIX pathname of a directory tree in which object files are to be compiled using \texttt{-xprofile=use}.

If you specify multiple instances of \texttt{-xprofile\_pathmap}, the compiler processes them in the order of their occurrence. Each \texttt{use\_prefix} specified by an instance of \texttt{-xprofile\_pathmap} is compared with the object file pathname until either a matching \texttt{use\_prefix} is identified or the last specified \texttt{use\_prefix} is found not to match the object file pathname.

3.4.163 \texttt{-xrecursive}

Allow routines without \texttt{RECURSIVE} attribute call themselves recursively.

Normally, only subprograms defined with the \texttt{RECURSIVE} attribute can call themselves recursively.

Compiling with \texttt{-xrecursive} enables subprograms to call themselves, even if they are not defined with the \texttt{RECURSIVE} attribute. But, unlike subroutines defined \texttt{RECURSIVE}, use of this flag does not cause local variables to be allocated on the stack by default. For local variables to have separate values in each recursive invocation of the subprogram, compile also with \texttt{-stackvar} to put local variables on the stack.

Indirect recursion (routine A calls routine B which then calls routine A) can give inconsistent results at optimization levels greater than \texttt{-xO2}. Compiling with the \texttt{-xrecursive} flag guarantees correctness with indirect recursion, even at higher optimization levels.

Compiling with \texttt{-xrecursive} can cause performance degradations.
3.4.164  -xreduction
Synonym for -reduction.

3.4.165  -xregs=r
Specify register usage.

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

Prefix the suboption with no% to disable the feature.

appl   (SPARC) Allow the compiler to generate code using the application registers as scratch registers. These are the g2, g3, and g4 registers on 32-bit processors, and g2, g3 on 64-bit processors.

float  (SPARC only) Allow the compiler to use the floating-point registers as scratch registers for integer values. This option has no effect on the compiler’s use of floating-point registers for floating-point values.

no%float Do not use the floating-point registers. With this option, a source program cannot contain any floating-point code.

frameptr (x86 only) Allow the compiler to use the frame-pointer register (%ebp on 32-bit x86, %rbp on 64-bit x86 processors) as an unallocated callee-saves register to improve program performance. -xregs=frameptr is ignored when also compiling with -xpg or -p.

The default is -xregs=appl,float on SPARC platforms, -xregs=appl,float,no%frameptr on x86.

It is strongly recommended that you compile code intended for shared libraries that will link with applications with -xregs=no%appl,float. At the very least, the shared library should explicitly document how it uses the application registers so that applications linking with those libraries know how to cope with the issue.

For example, an application using the registers in some global sense (such as using a register to point to some critical data structure) would need to know exactly how a library with code compiled without -xregs=no%appl is using the application registers in order to safely link with that library.
3.4.166  –xs

Allow debugging by dbx without object (.o) files.

With -xs, all debug information is copied into the executable file. If you move executables to another directory, then you can use dbx and ignore the object (.o) files. Use this option when you cannot retain the .o files.

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.

3.4.167  –xsafe=mem

(SPARC) Allow the compiler to assume that no memory protection violations occur.

Using this option allows the compiler to assume no memory–based traps occur. It grants permission to use the speculative load instruction on the SPARC V9 platforms.

This option is effective only when used with optimization level -05.

Caution – Because non-faulting loads do not cause a trap when a fault such as address misalignment or segmentation violation occurs, you should use this option only for programs in which such faults cannot occur. Because few programs incur memory-based traps, you can safely use this option for most programs. Do not use this option with programs that explicitly depend on memory-based traps to handle exceptional conditions.

3.4.168  –xsb

(Obsolete) Synonym for -sb.

3.4.169  –xsbfast

(Obsolete) Synonym for -sbfast.

3.4.170  –xspace

Do no optimizations that increase the code size.

Example: Do not unroll or parallelize loops if it increases code size.
3.4.171 \[-xtarget= t\]

Specify the target platform for the instruction set and optimization.

\(t\) must be one of: native, native64, generic, generic64, 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.

The actual expansion of \(-xtarget\) values can change from release to release. You can always determine the expansion that the compiler will use with the \(-dryrun\) flag:

```
demo% f95 -dryrun -xtarget=ultra4plus
### command line files and options (expanded):
### -dryrun -xarch=sparcvis
  -xcache=64/32/4:1:2048/64/4:2:32768/64/4:2 -xchip=ultra4plus
```

Note that the \(-xtarget\) expansion for a particular named platform might not be the same as \(-xtarget=native\) on that same platform.

3.4.171.1 Generic and Native Platforms

**native**
Optimize performance for the host platform (32-bits).

Expands to \(-m32 -xarch=native -xchip=native -xcache=native\)

**native64**
Obsolete. Use \(-xtarget=native -m64\) instead.

**generic**
Get the best performance for most 32-bit platforms.

This is the default and expands to:

\(-m32 -xarch=generic -xchip=generic -xcache=generic\)

**generic64**
Obsolete. Use \(-xtarget=generic -m64\) instead.

**platform-name**
Get the best performance for the specified platform listed below.

3.4.171.2 SPARC Platforms

The following table gives a list of the commonly used system platform names accepted by the compiler.
### 3.4 Options Reference

**Table 3-16  Expansions of Commonly Used **$\texttt{-xtarget}$** System Platforms**

<table>
<thead>
<tr>
<th>$\texttt{-xtarget}=\text{platform-name}$</th>
<th>$\texttt{-arch}$</th>
<th>$\texttt{-chip}$</th>
<th>$\texttt{-xcache}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sparc64vi</td>
<td>sparcfmaf</td>
<td>sparc64vi</td>
<td>128/64/2:5120/64/10</td>
</tr>
<tr>
<td>sparc64vii</td>
<td>sparcima</td>
<td>sparc64vii</td>
<td>64/64/2:5120/256/10</td>
</tr>
<tr>
<td>ultra</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra1/140</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra1/170</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra1/200</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra2</td>
<td>sparcvsis</td>
<td>ultra2</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra2/1170</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra2/1200</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:1024/64/1</td>
</tr>
<tr>
<td>ultra2/1300</td>
<td>sparcvsis</td>
<td>ultra2</td>
<td>16/32/1:2048/64/1</td>
</tr>
<tr>
<td>ultra2/2170</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra2/2200</td>
<td>sparcvsis</td>
<td>ultra</td>
<td>16/32/1:1024/64/1</td>
</tr>
<tr>
<td>ultra2/2300</td>
<td>sparcvsis</td>
<td>ultra2</td>
<td>16/32/1:2048/64/1</td>
</tr>
<tr>
<td>ultra2e</td>
<td>sparcvsis</td>
<td>ultra2e</td>
<td>16/32/1:256/64/4</td>
</tr>
<tr>
<td>ultra2i</td>
<td>sparcvsis</td>
<td>ultra2i</td>
<td>16/32/1:512/64/1</td>
</tr>
<tr>
<td>ultra3</td>
<td>sparcvsis</td>
<td>ultra3</td>
<td>64/32/4:8192/512/1</td>
</tr>
<tr>
<td>ultra3cu</td>
<td>sparcvsis</td>
<td>ultra3cu</td>
<td>64/32/4:8192/512/2</td>
</tr>
<tr>
<td>ultra3i</td>
<td>sparcvsis</td>
<td>ultra3i</td>
<td>64/32/4:1024/64/4</td>
</tr>
<tr>
<td>ultra4</td>
<td>sparcvsis</td>
<td>ultra4</td>
<td>64/32/4:8192/128/2</td>
</tr>
<tr>
<td>ultra4plus</td>
<td>sparcvsis</td>
<td>ultra4plus</td>
<td>64/32/4:1:2048/64/4/2:32768/64/4/2</td>
</tr>
<tr>
<td>ultraT1</td>
<td>sparc</td>
<td>ultraT1</td>
<td>8/16/4:4:3072/64/12/32</td>
</tr>
<tr>
<td>ultraT2</td>
<td>sparcvsis2</td>
<td>ultraT2</td>
<td>8/16/4:4096/64/16</td>
</tr>
<tr>
<td>ultraT2plus</td>
<td>sparcvsis2</td>
<td>ultraT2plus</td>
<td>8/16/4:4096/64/16</td>
</tr>
</tbody>
</table>

Compiling for a 64-bit Solaris OS on 64-bit enabled platforms is indicated by the **$\texttt{-m64}$** flag. If **$\texttt{-xtarget}$** is specified, **$\texttt{-m64}$** must appear after the **$\texttt{-xtarget}$** flag, as in:

```
$\texttt{-xtarget=ultra2 ... -m64}$
```

otherwise the default 32-bit memory model will be used.
3.4.171.3  x86 Platforms

The valid -xtarget platform names for x86 systems are:

generic, native, pentium, pentium_pro, pentium3, pentium4, woodcrest, penryn, nehalem, barcelona, and opteron.

TABLE 3–17  -xtarget Values on x86 Platforms

<table>
<thead>
<tr>
<th>-xtarget=</th>
<th>-xarch</th>
<th>-xchip</th>
<th>-xcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>generic</td>
<td>generic</td>
<td>generic</td>
<td>generic</td>
</tr>
<tr>
<td>opteron</td>
<td>sse2</td>
<td>opteron</td>
<td>64/64/2:1024/64/16</td>
</tr>
<tr>
<td>pentium</td>
<td>386</td>
<td>pentium</td>
<td>generic</td>
</tr>
<tr>
<td>pentium_pro</td>
<td>pentium_pro</td>
<td>pentium_pro</td>
<td>generic</td>
</tr>
<tr>
<td>pentium3</td>
<td>sse</td>
<td>pentium3</td>
<td>16/32/4:256/32/4</td>
</tr>
<tr>
<td>pentium4</td>
<td>sse2</td>
<td>pentium4</td>
<td>8/64/4:256/128/8</td>
</tr>
<tr>
<td>nehalem</td>
<td>sse4_2</td>
<td>nehalem</td>
<td>32/64/8:256/64/8:8192/64/16</td>
</tr>
<tr>
<td>penryn</td>
<td>sse4_1</td>
<td>penryn</td>
<td>2/64/8:4096/64/16</td>
</tr>
<tr>
<td>woodcrest</td>
<td>sse3</td>
<td>core2</td>
<td>32/64/8:4096/64/16</td>
</tr>
<tr>
<td>barcelona</td>
<td>amdsse4a</td>
<td>amdfam10</td>
<td>64/64/2:512/64/16</td>
</tr>
</tbody>
</table>

Compiling for 64-bit Solaris OS on 64-bit enabled x86 platform is indicated by the -m64 flag. For example, compiling with -xtarget=opteron is not necessary or sufficient. If -xtarget is specified, the -m64 option must appear after the -xtarget flag, as in:

-xtarget=opteron -m64

otherwise the compilation will be 32-bit x86.

3.4.172  –xtime

Synonym for -time.

3.4.173  –xtypemap=spec

Specify default data mappings.

This option provides a flexible way to specify the byte sizes for default data types. This option applies to both default-size variables and constants.
The specification string `spec` may contain any or all of the following in a comma-delimited list:

- `real:size`
- `double:size`
- `integer:size`

The allowable combinations on each platform are:

- `real:32`
- `real:64`
- `double:64`
- `double:128`
- `integer:16`
- `integer:32`
- `integer:64`

For example:

- `-xtypemap=real:64,double:64,integer:64`

maps both default `REAL` and `DOUBLE` to 8 bytes.

This option applies to all variables declared with default specifications (without explicit byte sizes), as in `REAL XYZ` (resulting in a 64-bit `XYZ`). Also, all single-precision `REAL` constants are promoted to `REAL*8`.

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.

### 3.4.174 `-xunroll=n`

Synonym for `-unroll=n`.

### 3.4.175 `-xvector=[[[no%]lib],[no%]simd,%none]]`

Enable automatic generation of calls to the vector library functions.

This option requires compiling with default rounding mode `-fround=nearest` when compiling with `-xvector`.

- `-xvector=lib` (Solaris platforms only) enables the compiler to transform math library calls within loops into single calls to the equivalent vector math routines when such transformations are possible. This could result in a performance improvement for loops with large loop counts.

- `-xvector=no%lib` disables this feature.

- `-xvector=simd` enables the compiler to use the native x86 SSE SIMD instructions to improve performance of certain loops. The compiler can only accept this switch if the target architecture supports SIMD instructions. For example, you must specify `-xarch=sse2 -m64` or `-xarch=generic64`. You must also specify an optimization level of `-xO3` or above as well as `-xdepend` with `-xvector=simd`. `-xvector=no%simd` disables this feature.
You will get better performance if you specify both -xvector=simd and -fsimple=2 than with -xvector=simd alone. However, your floating point results can be slightly different because -fsimple=2 allows reordering of floating-point operations.

The default is -xvector=%none. If you specify -xvector, but do not provide a sub-option, the compiler assumes -xvector=lib.

The compiler includes the libmvec libraries in the load step. If you specify -xvector=lib at compile time, you must also specify it at link time.

This option overrides previous instances so -xvector=%none overrides a previously specified -xvector=lib.

3.4.176 –ztext

Generate only pure libraries with no relocations.

The general purpose of -ztext is to 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% f95 -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% f95 -G -ztext -o MyLib -hMyLib a.f b.f x.o y.o

The options -ztext and -xprofile=collect should not be used together. While -ztext prohibits position-dependent symbol relocations in read-only storage, -xprofile=collect generates statically initialized, position-dependent symbol relocations in writable storage.

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.
This appendix shows some of the major features differences between standard Fortran and the Sun Fortran compiler, f95.

4.1 Source Language Features

The f95 compiler provides the following source language features and extensions to the Fortran standard.

4.1.1 Continuation Line Limits

f95 allows 999 continuation lines (1 initial and 999 continuation lines). Standard Fortran 95 allows 19 for fixed-form and 39 for free-form.

4.1.2 Fixed-Form Source Lines

In fixed-form source, lines can be longer than 72 characters, but everything beyond column 73 is ignored. Standard Fortran 95 only allows 72-character lines.

4.1.3 Tab Form

The f95 fixed-format 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.
4.1 Source Language Features

- 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 **f95** default maximum line length is 72 columns for fixed form and 132 for free form. Use the **-e** compiler option to extend the lines in fixed-form source to 132 columns.

Example: The tab form source on the left is treated as shown on the right.

```
! Uses of tabs
*CHARACTER *3 A = 'A'
*INTEGER B = 2
*REAL C = 3.0
*WRITE(*,9) A, B, C
9*FORMAT(1X, A3, 
    I1 I3,
    I2 F9.1 )
*IEND
```

In the example above, "*" stands for 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.

Tabs in **f95** force the rest of the line to be padded out to column 72. This may cause unexpected results if the tab appears within a character string that is continued onto the next line:

**Source file:**

```
*print *, "Tab on next line
*Ithis continuation line starts with a tab."
*Iend
```

**Running the code:**

```
Tab on next line this continuation line starts with a tab.
```

Tab formatting is also permitted with the **-f77** option.

4.1.4 Source Form Assumed

The source form assumed by **f95** depends on options, directives, and suffixes.

Files with a .f or .F suffix are assumed to be in fixed format. Files with a .f90, .f95, .F90, or .F95 suffix are assumed to be in free format.
### TABLE 4–1

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fixed</td>
<td>Interpret all source files as Fortran fixed form</td>
</tr>
<tr>
<td>-free</td>
<td>Interpret all source files as Fortran free form</td>
</tr>
</tbody>
</table>

If the `-free` or `-fixed` option is used, it overrides the file name suffix. If either a `!DIR$ FREE` or `!DIR$ FIXED` directive is used, it overrides the option and file name suffix.

#### 4.1.4 Mixing Forms

Some mixing of source forms is allowed.

- In the same `f95` command, some source files can be fixed form, some free.
- In the same file, free form can be mixed with fixed form by using `!DIR$ FREE` and `!DIR$ FIXED` directives.
- In the same program unit, tab form can be mixed with free or fixed form.

#### 4.1.5 Limits and Defaults

- A single Fortran program unit can define up to 65,535 derived types and 16,777,215 distinct constants.
- Names of variables and other objects can be up to 127 characters long. 31 is standard.

#### 4.2 Data Types

This section describes features and extensions to the Fortran data types.

##### 4.2.1 Boolean Type

`f95` supports constants and expressions of Boolean type. However, there are no Boolean variables or arrays, and there is no Boolean type statement.
4.2.1.1 Rules Governing Boolean Type

- For masking operations, 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, constant, or expression represents only two values, **.TRUE.** or **.FALSE.**
    A Boolean variable, constant, or expression can represent any binary value.
  - Logical entities are invalid in arithmetic, relational, or bitwise logical expressions.
    Boolean entities are valid in all three.

4.2.1.2 Alternate Forms of Boolean Constants

**f95** 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**

\[ ddddddB \]

- 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 (right most) digit specifies the content of the right most three bit positions (bits 29, 30, and 31).
- If less than 11 digits are present, the value is right-justified—it represents the right most 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

\( X'\text{ddd}' \) or \( X''\text{ddd}" \), 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 right most bits of a word: bits \( n \) through 31. The other bits are 0.

Hollerith

Accepted forms for Hollerith data are:

<table>
<thead>
<tr>
<th>( nH\ldots )</th>
<th>( '\ldots'H )</th>
<th>( &quot;\ldots&quot;H )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( nL\ldots )</td>
<td>( '\ldots'L )</td>
<td>( &quot;\ldots&quot;L )</td>
</tr>
<tr>
<td>( nR\ldots )</td>
<td>( '\ldots'R )</td>
<td>( &quot;\ldots&quot;R )</td>
</tr>
</tbody>
</table>

Above, \( "\ldots" \) 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.

<table>
<thead>
<tr>
<th>Boolean Constant</th>
<th>Internal Octal for 32-bit Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0B</td>
<td>000000000000</td>
</tr>
<tr>
<td>77740B</td>
<td>00000077740</td>
</tr>
<tr>
<td>X&quot;ABE&quot;</td>
<td>00000005276</td>
</tr>
<tr>
<td>X&quot;-340&quot;</td>
<td>37777776300</td>
</tr>
<tr>
<td>X'1 2 3'</td>
<td>00000000443</td>
</tr>
</tbody>
</table>
### Boolean Constant Internal Octal for 32-bit Word

<table>
<thead>
<tr>
<th>Boolean Constant</th>
<th>Internal Octal for 32-bit Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'FFFFFFFFFFFFFFF'</td>
<td>37777777777</td>
</tr>
</tbody>
</table>

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.

#### 4.2.1.3 Alternate Contexts of Boolean Constants

f95 allows BOZ constants in the places other than DATA statements.

<table>
<thead>
<tr>
<th>$B^{bb}$</th>
<th>$O^{oo}$</th>
<th>$Z^{zz}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B^{bbb}$</td>
<td>$O^{ooo}$</td>
<td>$Z^{zzz}$</td>
</tr>
</tbody>
</table>

If these are assigned to a real variable, no type conversion occurs.

Standard Fortran allows these only in DATA statements.

#### 4.2.2 Abbreviated Size Notation for Numeric Data Types

f95 allows the following nonstandard type declaration forms in declaration statements, function statements, and IMPLICIT statements. The form in column one is nonstandard Fortran, though in common use. The kind numbers in column two can vary by vendor.

<table>
<thead>
<tr>
<th>Table 4–2</th>
<th>Size Notation for Numeric Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonstandard</td>
<td>Declarator</td>
</tr>
<tr>
<td>INTEGER*1</td>
<td>INTEGER(KIND=1)</td>
</tr>
<tr>
<td>INTEGER*2</td>
<td>INTEGER(KIND=2)</td>
</tr>
<tr>
<td>INTEGER*4</td>
<td>INTEGER(KIND=4)</td>
</tr>
<tr>
<td>LOGICAL*1</td>
<td>LOGICAL(KIND=1)</td>
</tr>
<tr>
<td>LOGICAL*2</td>
<td>LOGICAL(KIND=2)</td>
</tr>
</tbody>
</table>
## 4.2.3 Size and Alignment of Data Types

Storage and alignment are always given in bytes. Values that can fit into a single byte are byte-aligned.

The size and alignment of types depends on various compiler options and platforms, and how variables are declared. The default maximum alignment in COMMON blocks is to 4-byte boundaries.

Default data alignment and storage allocation can be changed by compiling with special options, such as `-aligncommon`, `-f`, `-dalign`, `-dbl_align_all`, `-xmemalign`, and `-xtypemap`. The default descriptions in this manual assume that these options are not in force.

There is additional information in the *Fortran Programming Guide* regarding special cases of data types and alignment on certain platforms.

The following table summarizes the default size and alignment, ignoring other aspects of types and options.

<table>
<thead>
<tr>
<th>Nonstandard</th>
<th>Declarator</th>
<th>Short Form</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGICAL*4</td>
<td>LOGICAL(KIND=4)</td>
<td>LOGICAL(4)</td>
<td>Four-byte logicals</td>
</tr>
<tr>
<td>REAL*4</td>
<td>REAL(KIND=4)</td>
<td>REAL(4)</td>
<td>IEEE single-precision four-byte floating-point</td>
</tr>
<tr>
<td>REAL*8</td>
<td>REAL(KIND=8)</td>
<td>REAL(8)</td>
<td>IEEE double-precision eight-byte floating-point</td>
</tr>
<tr>
<td>REAL*16</td>
<td>REAL(KIND=16)</td>
<td>REAL(16)</td>
<td>IEEE quad-precision sixteen-byte floating-point</td>
</tr>
<tr>
<td>COMPLEX*8</td>
<td>COMPLEX(KIND=4)</td>
<td>COMPLEX(4)</td>
<td>Single-precision complex (four bytes each part)</td>
</tr>
<tr>
<td>COMPLEX*16</td>
<td>COMPLEX(KIND=8)</td>
<td>COMPLEX(8)</td>
<td>Double-precision complex (eight bytes each part)</td>
</tr>
<tr>
<td>COMPLEX*32</td>
<td>COMPLEX(KIND=16)</td>
<td>COMPLEX(16)</td>
<td>Quad-precision complex (sixteen bytes each part)</td>
</tr>
</tbody>
</table>
### 4.2 Data Types

#### TABLE 4-3  Default Data Sizes and Alignments (in Bytes)

<table>
<thead>
<tr>
<th>Fortran Data Type</th>
<th>Size</th>
<th>Default Alignment in COMMON</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CHARACTER X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CHARACTER*4 X</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>COMPLEX X</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>COMPLEX*8 X</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>DOUBLE COMPLEX X</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>COMPLEX*16 X</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>COMPLEX*32 X</td>
<td>32</td>
<td>8/16</td>
</tr>
<tr>
<td>DOUBLE PRECISION X</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>REAL X</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>REAL*4 X</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>REAL*8 X</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>REAL*16 X</td>
<td>16</td>
<td>8/16</td>
</tr>
<tr>
<td>INTEGER X</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>INTEGER*2 X</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INTEGER*4 X</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>INTEGER*8 X</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>LOGICAL X</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>LOGICAL*1 X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LOGICAL*2 X</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>LOGICAL*4 X</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>LOGICAL*8 X</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Note the following:

- **REAL*16** and **COMPLEX*32**: in 64-bit environments (compiling with `-m64`) the default alignment is on 16-byte (rather than 8-byte) boundaries, as indicated by 8/16 in the table. This data type is often referred to as **quad precision**.
- Arrays and structures align according to their elements or fields. An array aligns the same as the array element. A structure aligns the same as the field with the widest alignment.
Options `-f` or `-dalign` force alignment of all 8, 16, or 32-byte data onto 8-byte boundaries. Option `-dbl_align_all` causes all data to be aligned on 8-byte boundaries. Programs that depend on the use of these options may not be portable.

## 4.3 Cray Pointers

A *Cray pointer* is a variable whose value is the address of another entity, called the *pointee*.

**f95** supports Cray pointers; Standard Fortran 95 does not.

### 4.3.1 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:

- **pointer_name**
  
  Pointer to the corresponding `pointee_name`. `pointer_name` contains the address of `pointee_name`. Must be a scalar variable name (but not a derived type). Cannot be: a constant, a name of a structure, an array, or a function.

- **pointee_name**
  
  Pointee of the corresponding `pointer_name`

  Must be: a variable name, array declarator, or array name

- **array_spec**
  
  If `array_spec` is present, it must be explicit shape, (constant or non-constant bounds), or assumed-size.

For example, we can 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`.

We can also 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`. 
4.3.2 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.

4.3.3 Declaring Cray Pointers and Fortran 95 Pointers

Cray pointers are declared as follows:

```fortran
POINTER (pointer_name, pointee_name [array_spec])
```

Fortran 95 pointers are declared as follows:

```fortran
POINTER object_name
```

The two kinds of pointers cannot be mixed.

4.3.4 Features of Cray Pointers

- Whenever the pointee is referenced, _f95_ 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 on 32-bit processors, and two storage units on 64-bit processors.
- 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.

### 4.3.5 Restrictions on Cray Pointers

- `pointee_name` must not be a variable typed `CHARACTER*(*).`
- If `pointee_name` is an array declarator, it must be explicit shape, (constant or non-constant 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.

### 4.3.6 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.

### 4.3.7 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: \[ p = \text{LOC}(x) \]

Example: Use Cray pointers as described above.

```fortran
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 + 4000
  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

### 4.4 STRUCTURE and UNION (VAX Fortran)

To aid the migration of programs from *f77, f95* accepts VAX Fortran **STRUCTURE** and **UNION** statements, a precursor to the “derived types” in Fortran 95. For syntax details see the *FORTRAN 77 Language Reference* manual.

The field declarations within a **STRUCTURE** can be one of the following:

- A substructure — either another **STRUCTURE** declaration, or a record that has been previously defined.
- A **UNION** declaration.
- A **TYPE** declaration, which can include initial values.
A derived type having the `SEQUENCE` attribute. (This is particular to `f95` only.)

As with `f77`, a `POINTER` statement cannot be used as a field declaration.

`f95` also allows:
- Either ‘.’ or ‘s’ can be used as a structure field dereference symbol: `struct.field` or `struct%sfield`.
- Structures can appear in a formatted I/O statement.
- Structures can be initialized in a `PARAMETER` statement; the format is the same as a derived type initialization.
- Structures can appear as components in a derived type, but the derived type must be declared with the `SEQUENCE` attribute.

### 4.5 Unsigned Integers

The Fortran compiler accepts a new data type, `UNSIGNED`, as an extension to the language. Four `KIND` parameter values are accepted with `UNSIGNED`: 1, 2, 4, and 8, corresponding to 1-, 2-, 4-, and 8-byte unsigned integers, respectively.

The form of an unsigned integer constant is a digit-string followed by the upper or lower case letter U, optionally followed by an underscore and kind parameter. The following examples show the maximum values for unsigned integer constants:

```
255u_1
65535u_2
4294967295U_4
18446744073709551615U_8
```

Expressed without a kind parameter (`12345U`), the default is the same as for default integer. This is `U_4` but can be changed by the `xtypemap` option, which will change the kind type for default unsigned integers.

Declare an unsigned integer variable or array with the `UNSIGNED` type specifier:

```
UNSIGNED U
UNSIGNED(KIND=2) :: A
UNSIGNED*8 :: B
```
4.5.1 **Arithmetic Expressions**

- Binary operations, such as `+ - * /` cannot mix signed and unsigned operands. That is, `U*N` is illegal if `U` is declared `UNSIGNED`, and `N` is a signed `INTEGER`.
  - Use the `UNSIGNED` intrinsic function to combine mixed operands in a binary operation, as in `U*UNSIGNED(N)`
  - An exception is when one operand is an unsigned integer and the other is a signed integer constant expression with positive or zero value; the result is an unsigned integer.
  - The kind of the result of such a mixed expression is the largest kind of the operands.

Exponentiation of a signed value is signed while exponentiation of an unsigned value is unsigned.

- Unary minus of an unsigned value is unsigned.
- Unsigned operands may mix freely with real, complex operands. (Unsigned operands cannot be mixed with interval operands.)

4.5.2 **Relational Expressions**

Signed and unsigned integer operands may be compared using intrinsic relational operations. The result is based on the unaltered value of the operands.

4.5.3 **Control Constructs**

- The `CASE` construct accepts unsigned integers as case-expressions.
- Unsigned integers are not permitted as DO loop control variables, or in arithmetic IF control expressions.

4.5.4 **Input/Output Constructs**

- Unsigned integers can be read and written using the I, B, O, and Z edit descriptors.
- They can also be read and written using list-directed and namelist I/O. The written form of an unsigned integer under list-directed or namelist I/O is the same as is used for positive signed integers.
- Unsigned integers can also be read or written using unformatted I/O.
4.5.5 Intrinsic Functions

- Unsigned integers are allowed in intrinsic functions, except for \texttt{SIGN} and \texttt{ABS}.
- A new intrinsic function, \texttt{UNSIGNED}, is analogous to \texttt{INT} but produces a result of unsigned type. The form is
  \begin{verbatim}
  UNSIGNED(v [,kind])
  \end{verbatim}
- Another new intrinsic function, \texttt{SELECTED_UNSIGNED_KIND( var)}, returns the kind parameter for \texttt{var}.
- Intrinsic functions do not allow both signed and unsigned integer operands, except for the \texttt{MAX} and \texttt{MIN} functions, which allow both signed and unsigned integer operands only if there is at least one operand of \texttt{REAL} type.
- Unsigned arrays cannot appear as arguments to array intrinsic functions.

4.6 Fortran 2003 Features

A number of new features in the Fortran 2003 standard appear in this release of the \texttt{f95} compiler. For details, refer to the Fortran 2003 standard.

4.6.1 Interoperability with C Functions

The new standard for Fortran provides:

- a means of referencing C language procedures and, conversely, a means of specifying that a Fortran subprogram can be referenced from a C function, and
- a means of declaring global variables that are linked with external C variables

The \texttt{ISO_C_BINDING} module provides access to named constants that are kind type parameters representing data that is compatible with C types.

The standard also introduces the \texttt{BIND(C)} attribute. A Fortran derived type is interoperable with C if it has the \texttt{BIND} attribute.

This release of the Fortran compiler implements these features as described in the chapter 15 of the standard. Fortran also provides facilities for defining derived types and enumerations that correspond to C types, as described in chapter 4 of the standard.
4.6.2 IEEE Floating-Point Exception Handling

New intrinsic modules \texttt{IEEE_ARITHMETIC}, and \texttt{IEEE_FEATURES} provide support for exceptions and IEEE arithmetic in the Fortran language. Full support of these features is provided by:

\begin{verbatim}
USE, INTRINSIC :: IEEE_ARITHMETIC
USE, INTRINSIC :: IEEE_FEATURES
\end{verbatim}

The \texttt{INTRINSIC} keyword is new in Fortran 2003. These modules define a set of derived types, constants, rounding modes, inquiry functions, elemental functions, kind functions, and elemental and non-elemental subroutines. The details are contained in Chapter 14 of the Fortran 2003 standard.

4.6.3 Command-Line Argument Intrinsics

The Fortran 2003 standard introduces three new intrinsics for processing command-line arguments and environment variables. These are:

- \texttt{GET_COMMAND(command, length, status)}
  
  Returns in \texttt{command} the entire command line that invoked the program.

- \texttt{GET_COMMAND_ARGUMENT(number, value, length, status)}
  
  Returns a command-line argument in \texttt{value}.

- \texttt{GET_ENVIRONMENT_VARIABLE(name, value, length, status, trim_name)}
  
  Return the value of an environment variable.

4.6.4 PROTECTED Attribute

The Fortran compiler now accepts the Fortran 2003 \texttt{PROTECTED} attribute. \texttt{PROTECTED} imposes limitations on the usage of module entities. Objects with the \texttt{PROTECTED} attribute are only definable within the module that declares them.

4.6.5 Fortran 2003 Asynchronous I/O

The compiler recognizes the \texttt{ASYNCRONOUS} specifier on I/O statements:

\begin{verbatim}
ASYNCRONOUS=[‘YES’ | ‘NO’]
\end{verbatim}

This syntax is as proposed in the Fortran 2003 standard, Chapter 9. In combination with the \texttt{WAIT} statement it allows the programmer to specify I/O processes that may be overlapped with
computation. While the compiler recognizes `ASYNCHRONOUS = 'YES'`, the standard does not require actual asynchronous I/O. In this release of the compiler, I/O is always synchronous.

4.6.6 **Extended ALLOCATABLE Attribute**

Fortran 2003 extends the data entities allowed for the `ALLOCATABLE` attribute. Previously this attribute was limited to locally stored array variables. It is now allowed with:

- array components of structures
- dummy arrays
- array function results

Allocatable entities remain forbidden in all places where they may be storage-associated: `COMMON` blocks and `EQUIVALENCE` statements. Allocatable array components may appear in `SEQUENCE` types, but objects of such types are then prohibited from `COMMON` and `EQUIVALENCE`.

4.6.7 **VALUE Attribute**

The f95 compiler accepts the Fortran 2003 `VALUE` type declaration attribute.

Specifying a subprogram dummy input argument with this attribute indicates that the actual argument is passed “by value”. The following example demonstrates the use of the `VALUE` attribute with a C main program calling a Fortran subprogram with a literal value as an argument:

C code:
```c
#include <stdlib.h>
int main(int ac, char *av[]) {
    to_fortran(2);
}
```

Fortran code:
```fortran
subroutine to_fortran(i)
    integer, value :: i
    print *, i
end
```
4.6.8 Fortran 2003 Stream I/O

The Fortran 2003 standard defines a new “stream” I/O scheme. Stream I/O access treats a data file as a continuous sequence of bytes, addressable by a positive integer starting from 1. The data file can be connected for either formatted or unformatted access.

Declare a stream I/O file with the ACCESS= 'STREAM' specifier on the OPEN statement. File positioning to a byte address requires a POS= scalar_integer_expression specifier on a READ or WRITE statement. The INQUIRE statement accepts ACCESS= 'STREAM', a specifier STREAM= scalar_character_variable, and POS= scalar_integer_variable.

4.6.9 Fortran 2003 Formatted I/O Features

Three new Fortran 2003 formatted I/O specifiers have been implemented in f95. They may appear on OPEN, READ, WRITE, PRINT, and INQUIRE statements:

- **DECIMAL=[‘POINT’ | ‘COMMA’]**
  Change the default decimal editing mode. The default uses a period to separate the whole number and decimal parts of a floating-point number formatted with D, E, EN, ES, F, and G editing. ‘COMMA’ changes the default to use comma instead of a period, to print, for example, 123,456. The default is ‘POINT’, which uses a period, to print, for example, 123.456.

- **ROUND=[‘PROCESSOR_DEFINED’ | ‘COMPATIBLE’]**
  Set the default rounding mode for formatted I/O D, E, EN, ES, F, and G editing. With ‘COMPATIBLE’, the value resulting from data conversion is the one closer to the two nearest representations, or the value away from zero if the value is halfway between them. With ‘PROCESSOR_DEFINED’, the rounding mode is dependent on the processor’s default mode, and is the compiler default if ROUND is not specified.
  As an example, WRITE(*, '(f11.4)') 0.11115 prints 0.1111 in default mode, and 0.1112 in ‘COMPATIBLE’ mode.

- **IOMSG=character-variable**
  Returns an error message as a string in the specified character variable. This is the same error message that would appear on standard output. Users should allocate a character buffer large enough to hold the longest message. (CHARACTER*256 should be sufficient.)

When used in INQUIRE statements, these specifiers declare a character variable for returning the current values.

New edit descriptors DP, DC, RP, and RC change the defaults within a single FORMAT statement to decimal point, decimal comma, processor-defined rounding, and compatible rounding respectively. For example:

WRITE(*, '(i5,DC,F10.3)') N, W
prints a comma instead of a period in the F10.3 output item.

See also the -iorounding compiler command-line option for changing the floating-point rounding modes on formatted I/O. ("3.4.46 -iorounding=[compatible|processor-defined]") on page 72.)

4.6.10 Fortran 2003 IMPORT Statement

The IMPORT statement specified entities in the host scoping unit that are accessible by host association. It is allowed only in an interface body.

4.6.11 Fortran 2003 FLUSH I/O Statement

The f95 compiler accepts the Fortran 2003 FLUSH statement. The FLUSH statement makes data written to an external file available to other processes, or causes data placed in an external file by means other than Fortran to be available to a READ statement.

4.7 Additional I/O Extensions

The section describes extensions to Fortran 95 Input/Output handling that are accepted by the f95 compiler that are not part of the Fortran 2003 standard. Some are I/O extensions that appeared in the Fortran 77 compiler, f77, and are now part of the Fortran compiler.

4.7.1 I/O Error Handling Routines

Two new functions enable the user to specify their own error handling routine for formatted input on a logical unit. When a formatting error is detected, the runtime I/O library calls the specified user-supplied handler routine with data pointing at the character in the input line causing the error. The handler routine can supply a new character and allow the I/O operation to continue at the point where the error was detected using the new character; or take the default Fortran error handling.

The new routines, SET_IO_ERR_HANDLER(3f) and GET_IO_ERR_HANDLER(3f), are module subroutines and require USE SUN_IO_HANDLERS in the routine that calls them. See the man pages for these routines for details.
4.7.2 Variable Format Expressions

Fortran 77 allowed any integer constant in a format to be replaced by an arbitrary expression enclosed in angle brackets:

```fortran
1 FORMAT( ... < expr > ... )
```

Variable format expressions cannot appear as the \( n \) in an \( nH \ldots \) edit descriptor, in a `FORMAT` statement that is referenced by an `ASSIGN` statement, or in a `FORMAT` statement within a parallel region.

This feature is enabled natively in f95, and does not require the `-f77` compatibility option flag.

4.7.3 NAMELIST Input Format

- The group name may be preceded by `$` or `&` on input. The `&` is the only form accepted by the Fortran 95 standard, and is what is written by `NAMELIST` output.
- Accepts `$` as the symbol terminating input except if the last data item in the group is `CHARACTER` data, in which case the `$` is treated as input data.
- Allows `NAMELIST` input to start in the first column of a record.

4.7.4 Binary Unformatted I/O

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.

- `WRITE` statement: Data is written to the file in binary, with as many bytes transferred as specified by the output list.
- `READ` statement: Data is read into the variables on the input list, transferring as many bytes as required by the list. Because there are no record marks on the file, there will be no "end-of-record" error detection. The only errors detected are "end-of-file" or abnormal system errors.
- `INQUIRE` statement: `INQUIRE` on a file opened with `FORM='BINARY'` returns:
  - `FORM='BINARY'` `ACCESS='SEQUENTIAL'` `DIRECT='NO'` `FORMATTED='NO'` `UNFORMATTED='YES'` `RECL= AND NEXTREC=` are undefined
- `ENDFILE` statement: Truncates file at current position, as usual.
4.7.5 Miscellaneous I/O Extensions

- Recursive I/O possible on different units (this is because the f95 I/O library is "MT-Warm").
- **RECL=2147483646** \((2^{31} - 2)\) is the default record length on sequential formatted, list directed, and namelist output.
- **ENCODE** and **DECODE** are recognized and implemented as described in the FORTRAN 77 Language Reference Manual.
- Non-advancing I/O is enabled with **ADVANCE='NO'**, as in:
  ```fortran
  write(*,'(a)',ADVANCE='NO') 'n= ' read(*,*) n
  ```

4.8 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.

A complete summary of Fortran directives appears in Table C-1.

4.8.1 Form of Special f95 Directive Lines

f95 recognizes its own special directives in addition to those described in "1.9 Command-Line Help" on page 21. These have the following syntax:

```fortran
!DIR$ d1, d2, ...
```

4.8.1.1 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.
4.8.1.2 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.

4.8.2 FIXED and FREE Directives

These directives specify the source form of lines following the directive line.

4.8.2.1 Scope

They apply to the rest of the file in which they appear, or until the next FREE or FIXED directive is encountered.

4.8.2.2 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.

4.8.2.3 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
4.8.3 Parallelization Directives

A parallelization directive is a special comment that directs the compiler to attempt to parallelize the next DO loop. These are summarized in Appendix D and described in the chapter on parallelization in the Fortran Programming Guide. Both Sun and Cray style parallelization directives are now deprecated as obsolete. The OpenMP Fortran API directives and parallelization model is preferred. OpenMP parallelization is described in the OpenMP API User's Guide.

4.9 Module Files

Compiling a file containing a Fortran 95 MODULE generates a module interface 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.

Compilation also generates a .o module implementation object file for the source file containing the MODULE statements. Link with the module implementation object file along with the all other object files to create an executable.

The compiler creates module interface files and implementation object files in the directory specified by the -moddir=dir flag or the MODDIR environment variable. If not specified, the compiler writes .mod files in the current working directory.

The compiler looks in the current working directory for the interface files when compiling USE modulename statements. The -Mpath option allows you to give the compiler an additional path to search. Module implementation object files must be listed explicitly on the command line for the link step.

Typically, programmers define one MODULE per file and assign the same name to the MODULE and the source file containing it. However, this is not a requirement.

In this example, all the files are compiled at once. The module source files appear first before their use in the main program.

demo% cat mod_one.f90
MODULE one
   ...
END MODULE

demo% cat mod_two.f90
MODULE two
4.9 Module Files

... 
END MODULE

demo% cat main.f90
USE one
USE two
...
END

demo% f95 -o main mod_one.f90 mod_two.f90 main.f90

Compilation creates the files:

main main.o one.mod mod_one.o two.mod mod_two.o

The next example compiles each unit separately and links them together.

demo% f95 -c mod_one.f90 mod_two.f90
demo% f95 -c main.f90
demo% f95 -o main main.o mod_one.o mod_two.o

When compiling main.f90, the compiler searches the current directory for one.mod and two.mod. These must be compiled before compiling any files that reference the modules on USE statements. The link step requires the module implementation object files mod_one.o and mod_two.o appear along with all other object files to create the executable.

4.9.1 Searching for Modules

With the release of the version 7.0 of the Fortran compiler, .mod files can be stored into an archive (.a) file. An archive must be explicitly specified in a -Mpath flag on the command line for it to be searched for modules. The compiler does not search archive files by default.

Only .mod files with the same names that appear on USE statements will be searched. For example, the Fortran statement USE mymod causes the compiler to search for the module file mymod.mod by default.

While searching, the compiler gives higher priority to the directory where the module files are being written. This can be controlled by the -moddir=dir option flag and the MODDIR environment variable. This implies that if only the -Mpath option is specified, the current directory will be searched for modules first, before the directories and files listed on the -M flag.

4.9.2 The -use=list Option Flag

The -use=list flag forces one or more implicit USE statements into each subprogram or module subprogram compiled with this flag. By using the flag, it is not necessary to modify source programs when a module or module file is required for some feature of a library or application.
4.10 Intrinsics

**The fdumpmod Command**

Use the `fdumpmod(1)` command to display information about the contents of a compiled module information file.

```
demo% fdumpmod x.mod group.mod
x 1.0 v8,i4,r4,d8,n16,a4 x.mod
  group 1.0 v8,i4,r4,d8,n16,a4 group.mod
```

The `fdumpmod` command will display information about modules in a single `.mod` file, files formed by concatenating `.mod` files, and in `.a` archives of `.mod` files. The display includes the name of the module, a version number, the target architecture, and flags indicating compilation options with which the module is compatible. See the `fdumpmod(1)` man page for details.

4.10 Intrinsics

`f95` supports some intrinsic procedures that are extensions beyond the standard.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Function Type</th>
<th>Argument Types</th>
<th>Arguments</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COT</td>
<td>Cotangent</td>
<td>real</td>
<td>real</td>
<td>(X=x)</td>
<td>P, E</td>
</tr>
<tr>
<td>DDIM</td>
<td>Positive difference</td>
<td>double precision</td>
<td>double precision</td>
<td>(X=x, Y=y)</td>
<td>P, E</td>
</tr>
<tr>
<td>LEADZ</td>
<td>Get the number of leading 0 bits</td>
<td>integer</td>
<td>Boolean, integer, real, or pointer</td>
<td>(I=i)</td>
<td>NP, I</td>
</tr>
<tr>
<td>POPCNT</td>
<td>Get the number of set bits</td>
<td>integer</td>
<td>Boolean, integer, real, or pointer</td>
<td>(I=i)</td>
<td>NP, I</td>
</tr>
<tr>
<td>POPPAR</td>
<td>Calculate bit population parity</td>
<td>integer</td>
<td>Boolean, integer, real, or pointer</td>
<td>(X=x)</td>
<td>NP, I</td>
</tr>
</tbody>
</table>

**Notes:** 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: `f95` generates inline code for the intrinsic procedure.

See the *Fortran Library Reference* for a more complete discussion of intrinsics, including those from Fortran 77 that are recognized by the Fortran compiler.
4.11 Forward Compatibility

Future releases of f95 are intended to be source code compatible with this release.

Module information files generated by this release of f95 are not guaranteed to be compatible with future releases.

4.12 Mixing Languages

Routines written in C can be combined with Fortran programs, since these languages have common calling conventions. See the C/Fortran Interface chapter in the Fortran Programming Guide for details on how to interoperate between C and Fortran routines.
FORTRAN 77 Compatibility: Migrating to Sun Studio Fortran

The Fortran compiler, f95, will compile most legacy FORTRAN 77 programs, including programs utilizing non-standard extensions previously compiled by the f77 compiler.

f95 will accept many of these FORTRAN 77 features directly. Others require compiling in FORTRAN 77 compatibility mode (f95 -f77).

This chapter describes the FORTRAN 77 features accepted by f95, and lists those f77 features that are incompatible with f95. For details on any of the non-standard FORTRAN 77 extensions that were accepted by the f77 compiler, see the FORTRAN 77 Language Reference manual at http://docs.sun.com/source/806-3594/index.html.

See Chapter Chapter 4, “Sun Studio Fortran Features and Differences,” for other extensions to the Fortran language accepted by the f95 compiler.

f95 will compile standard-conforming FORTRAN 77 programs. To ensure continued portability, programs utilizing non-standard FORTRAN 77 features should migrate to standard-conforming Fortran 95/2003. Compiling with the -ansi option will flag all non-standard usages in your program.

5.1 Compatible f77 Features

f95 accepts the following non-standard features of the FORTRAN 77 compiler, f77, either directly or when compiling in -f77 compatibility mode:

Source Format

- Continuation lines can start with "&" in column 1. [-f77=misc]
- The first line in an include file can be a continuation line. [-f77=misc]
- Use f77 tab-format. [-f77=tab]
- Tab-formatting can extend source lines beyond column 72. [-f77=tab]
f95 tab-formatting will not pad character strings to column 72 if they extend over a continuation line. [-f77]

I/O:
- You can open a file with ACCESS='APPEND' in Fortran 95.
- List-directed output uses formats similar to the f77 compiler. [-f77=output]
- f95 allows BACKSPACE on a direct-access file, but not ENDFILE.
- f95 allows implicit field-width specifications in format edit descriptors. For example, FORMAT(I) is allowed.
- f95 will recognize f77 escape sequences (for example, \n \t \') in output formats. [-f77=backslash.]
- f95 recognizes FILEOPT= in OPEN statements.
- f95 allows SCRATCH files to be closed with STATUS='KEEP' [-f77]. When the program exits, the scratch file is not deleted. SCRATCH files can also be opened with FILE=name when compiled with -f77.
- Direct I/O is permitted on internal files. [-f77]
- f95 recognizes FORTRAN 77 format edit descriptors A, $, and SU. [-f77]
- FORM= ‘PRINT’ can appear on OPEN statements. [-f77]
- f95 recognizes the legacy FORTRAN input/output statements ACCEPT and TYPE.
- Compile with -f77=output to write FORTRAN 77 style NAMELIST output.
- A READ with only ERR= specified (no IOSTAT= or END= branches) treats the ERR= branch as an END= when an EOF is detected. [-f77]
- VMS Fortran NAME=’filename’ is accepted on OPEN statements. [-f77]
- f95 accepts an extra comma after READ() or WRITE(). [-f77]
- END= branch can appear on direct access READ with REC=. [-f77=input]
- Allow format edit descriptor Ew.d.e and treat it as Ew.d.Ee. [-f77]
- Character strings can be used in the FORMAT of an input statement. [-f77=input]
- IOSTAT= specifier can appear in ENCODE/DECODE statements.
- List-directed I/O is allowed with ENCODE/DECODE statements.
- Asterisk (*) can be used to stand in for STIDN and STDOUT when used as a logical unit in an I/O statement.
- Arrays can appear in the FMT= specifier. [-f77=misc]
- PRINT statement accepts namelist group names. [-f77=output]
- The compiler accepts redundant commas in FORMAT statements.
- While performing NAMELIST input, entering a question mark (?) responds with the name of the namelist group being read. [-f77=input]
Data Types, Declarations, and Usage:

- In a program unit, the `IMPLICIT` statement may follow any other declarative statement in the unit.
- `f95` accepts the `IMPLICIT UNDEFINED` statement.
- `f95` accepts the `AUTOMATIC` statement, a FORTRAN 77 extension.
- `f95` accepts the `STATIC` statement and treats it like a `SAVE` statement.
- `f95` accepts VAX `STRUCTURE`, `UNION`, and `MAP` statements. (See "4.4 STRUCTURE and UNION (VAX Fortran)" on page 152)
- `LOGICAL` and `INTEGER` variables can be used interchangeably. [-f77=logical]
- `INTEGER` variables can appear in conditional expressions, such as `DO WHILE`. [-f77=logical]
- Cray pointers can appear in calls to intrinsic functions.
- `f95` will accept data initializations using slashes on type declarations. For example: `REAL MHW/100.101/, ICOMX/32.223/`
- `f95` allows assigning Cray character pointers to non-pointer variables and to other Cray pointers that are not character pointers.
- `f95` allows the same Cray pointer to point to items of different type sizes (for example, REAL*8 and INTEGER*4). A Cray pointer can be declared `INTEGER` in the same program unit where it is declared as a `POINTER`. The `INTEGER` declaration is ignored. [-f77=misc]
- A Cray pointer can be used in division and multiplication operations. [-f77=misc]
- Variables in an `ASSIGN` statement can be of type `INTEGER*2`. [-f77=misc]
- Expressions in alternate `RETURN` statements can be non-integer type. [-f77=misc]
- Variables with the `SAVE` attribute can be equivalenced to an element of a `COMMON` block.
- Initializers for the same array can be of different types. Example: `REAL*8 ARR(5) /12.3 1, 3, 5.00, 9/`
- Type declarations for namelist items can follow the `NAMELIST` statement.
- `f95` accepts the `BYTE` data type.
- `f95` allows non-integers to be used as array subscripts. [-f77=subscript]
- `f95` allows relational operators `.EQ.` and `.NE.` to be used with logical operands. [-f77=logical]
- `f95` will accept the legacy `f77 VIRTUAL` statement, and treats it as a `DIMENSION` statement.
- Different data structures can be equivalenced in a manner that is compatible with the `f77` compiler. [-f77=misc]
- Like the `f77` compiler, `f95` allows many intrinsics to appear in initialization expressions on `PARAMETER` statements.
- `f95` allows assignment of an integer value to `CHARACTER*1` variables. [-f77=misc]
BOZ constants can be used as exponents. [-f77=misc]

BOZ constants can be assigned to character variables. For example:

```fortran
character*8 ch
ch = "12345678"X
```

BOZ constants can be used as arguments to intrinsic function calls. [-f77=misc]

A character variable can be initialized with an integer value in a DATA statement. The first character in the variable is set to the integer value and the rest of the string, if longer than one character, is blank-filled.

An integer array of hollerith characters can be used as a format descriptor. [-f77].

Constant folding will not be done at compile time if it results in a floating-point exception. [-f77=misc]

When compiling with -f77=misc, f95 will automatically promote a REAL constant to the appropriate kind (REAL*8 or REAL*16) in assignments, data, and parameter statements, in the manner of the f77 compiler. [-f77=misc]

Equivalenced variables are allowed on an assigned GOTO. [-f77]

Non-constant character expressions can be assigned to numeric variables. [-f77=misc]

Compiling with -f77=misc allows *kind after the variable name in type declarations. [-f77=misc]. For example:

```fortran
PARAMETER (N=2) INTEGER*(N+2) K
```

A subroutine can call itself recursively without declaring itself with a RECURSIVE keyword. [-f77=misc] However, programs that perform indirect recursion (routine A calls routine B which then calls routine A) should also be compiled with the -xrecursive flag to work correctly.

A subroutine with alternate returns can be called even when the dummy argument list does not have an alternate return list.
Compiling with -f77=misc allows statement functions to be defined with arguments typed other than INTEGER or REAL, and actual arguments will be converted to the type defined by the statement function. [-f77=misc]

Allow null actual arguments. For example: CALL FOO(I,,J) has two null arguments between the first I and the final J argument.

-f95 treats a call to the function %LOC() as a call to LOC(). [-f77=misc]

Allow unary plus and unary minus after another operator such as ** or *.

Allow a second argument with the CMPLX() intrinsic function even when the first argument is a COMPLEX type. In this case, the real part of the first argument is used. [-f77=misc]

Allow the argument to the CHAR() intrinsic function to exceed 255 with only a warning, not an error. [-f77=misc]

Allow negative shift counts with only a warning, not an error.

Search for an INCLUDE file in the current directory as well as those specified in the -I option. [-f77=misc]

Allow consecutive .NOT. operations, such as .NOT..NOT..NOT.(I.EQ.J). [-f77=misc]

Miscellaneous

The f95 compiler normally does not issue progress messages to standard out. The f77 compiler did issue progress messages, displaying the names of the routines it was compiling. This convention is retained when compiling with the -f77 compatibility flag.

Programs compiled by the f77 compiler did not trap on arithmetic exceptions, and automatically called ieee_retrospective on exit to report on any exceptions that may have occurred during execution. Compiling with the -f77 flag mimics this behavior of the f77 compiler. By default, the f95 compiler traps on the first arithmetic exception encountered and does not call ieee_retrospective.

The f77 compiler treated a REAL*4 constant as if it had higher precision in contexts where higher precision was needed. When compiling with the -f77 flag, the f95 compiler allows a REAL*4 constant to have double or quad precision when the constant is used with a double or quad precision operand, respectively.

Allow the DO loop variable to be redefined within the loop. [-f77=misc]

Display the names of program units being compiled. [-f77=misc]

Allow the types of the variables used in a DIMENSION statement to be declared after the DIMENSION statement. Example:

SUBROUTINE FOO(ARR,G)
DIMENSION ARR(G)
INTEGER G
RETURN
END
5.2 Incompatibility Issues

The following lists known incompatibilities that arise when compiling and testing legacy f77 programs with this release of f95. These are due to either missing comparable features in f95, or differences in behavior. These items are non-standard extensions to Fortran 77 supported in f77 but not in f95.

Source Format

- An ANSI warning is given for names longer than 6 characters when the -f77 option is specified.

I/O:

- f95 does not allow ENDFILE on a direct-access file.
- f95 does not recognize the 'n form for specifying a record number in direct access I/O: READ (2 '13) X, Y, Z
- f95 does not recognize the legacy f77 “R” format edit descriptor.
- f95 does not allow the DISP= specifier in a CLOSE statement.
- Bit constants are not allowed on a WRITE statement.
- Fortran 95 NAMELIST does not allow arrays and character strings with variable lengths.
- Opening a direct access file with RECL=1 cannot be used as a “stream” file. Use FORMAT=’STREAM’ instead.
- Fortran 95 reports illegal I/O specifiers as errors. f77 gave only warnings.

Data Types, Declarations, and Usage:

- f95 allows only 7 array subscripts; f77 allowed 20.
- f95 does not allow non-constants in PARAMETER statements.
- Integer values cannot be used in the initializer of a CHARACTER type declaration.
- The REAL() intrinsic returns the real part of a complex argument instead of converting the argument to REAL*4. This gives different results when the argument is DOUBLE COMPLEX or COMPLEX*32
- Fortran 95 will not allow array elements in boundary expressions before the array is declared. For example:

  subroutine s(i1,i2)
  integer i1(i2(1):10)
  dimension i2(10)
...ERROR: "I2" has been used as a function, therefore it must not be declared with the explicit-shape DIMENSION attribute.

end

Programs, Subroutines, Functions, Statements:

- The maximum length for names is 127 characters.

Command-line Options:

- f95 does not recognize the f77 compiler options -dbl -oldstruct -i2 -i4 and some suboptions of -vax.

f77 Library Routines Not Supported by f95:

- The POSIX library.
- The IOINIT() library routine.
- The tape I/O routines open, tclose, twrite, tread, trewin, tskipf, tstate.
- start_iostats and end_iostats library routines.
- f77_init() function.
- f95 does not allow the IEEE_RETROSPECTIVE subroutine to be bypassed by defining the user’s own routine with the same name.

5.3 Linking With f77-Compiled Routines

- To mix f77 and f95 object binaries, link with f95 compile with the -xlang=f77 option. Perform the link step with f95 even if the main program is an f77 program.
- Example: Compiling an f95 main program with an f77 object file.

```f95
CHARACTER*74 :: c = 'This is a test.'
    CALL echo1( c )
END
```

```f95
f95 -xlang=f77 m.f95 sub77.o
a.out
```

- The FORTRAN 77 library and intrinsics are available to f95 programs and are listed in the Fortran Library Reference Manual.
- Example: f95 main calls a routine from the FORTRAN 77 library.

```f95
REAL e, dtime, t(2)
```
5.3.1 Fortran Intrinsics

The Fortran standard supports intrinsic functions that FORTRAN 77 did not have. The full set of Fortran intrinsics, including non-standard intrinsics, appears in the Fortran Library Reference manual.

If you use any of the intrinsic names listed in the Fortran Library Reference as a function name in your program, you must add an EXTERNAL statement for f95 to use your routine rather than the intrinsic one.

The Fortran Library Reference also lists all the intrinsics recognized by earlier releases of the f77 compiler. The f95 compiler recognizes these names as intrinsics as well.

Compiling with -f77=intrinsics limits the compiler’s recognition of intrinsic functions to just those that were known to the f77 compiler, ignoring the Fortran intrinsics.

5.4 Additional Notes About Migrating to the f95 Compiler

- The floatingpoint.h header file replaces f77_floatingpoint.h, and should be used in source programs as follows:

```c
#include "floatingpoint.h"
```

- Header file references of the form f77/ilename should be changed to remove the f77/directory path.

- Some programs utilizing non-standard aliasing techniques (by overindexing arrays, or by overlapping Cray or Fortran pointers) may benefit by compiling with the appropriate -xalias flag. See “3.4.108 -xalias[=keywords]” on page 93. This is discussed with examples in the chapter on porting “dusty deck” programs in the Fortran Programming Guide.
This appendix describes the error messages generated by the Fortran runtime I/O library and operating system.

A.1 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:

```fortran
integer system, status
status = system("cp afile bfile")
print*, "status = ", status
end
```

The following message is displayed:

```plaintext
cp: cannot access afile
status = 512
```

A.2 f95 Runtime I/O Error Messages

The f95 I/O library issues diagnostic messages when errors are detected at runtime. Here is an example, compiled and run with f95:

```bash
demo% cat wf.f
WRITE( 6 ) 1
END
demo% f95 -o wf wf.f
```
Because the f95 message contains references to the originating source code filename and line number, application developers should consider using the ERR= clause in I/O statements to softly trap runtime I/O errors.

Table A–1 lists the runtime I/O messages issued by f95.

<table>
<thead>
<tr>
<th>Error</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>format error</td>
</tr>
<tr>
<td>1001</td>
<td>illegal unit number</td>
</tr>
<tr>
<td>1002</td>
<td>formatted I/O on unformatted unit</td>
</tr>
<tr>
<td>1003</td>
<td>unformatted I/O on formatted unit</td>
</tr>
<tr>
<td>1004</td>
<td>direct-access I/O on sequential-access unit</td>
</tr>
<tr>
<td>1005</td>
<td>sequential-access I/O on direct-access unit</td>
</tr>
<tr>
<td>1006</td>
<td>device does not support BACKSPACE</td>
</tr>
<tr>
<td>1007</td>
<td>off beginning of record</td>
</tr>
<tr>
<td>1008</td>
<td>can't stat file</td>
</tr>
<tr>
<td>1009</td>
<td>no * after repeat count</td>
</tr>
<tr>
<td>1010</td>
<td>record too long</td>
</tr>
<tr>
<td>1011</td>
<td>truncation failed</td>
</tr>
<tr>
<td>1012</td>
<td>incomprehensible list input</td>
</tr>
<tr>
<td>1013</td>
<td>out of free space</td>
</tr>
<tr>
<td>1014</td>
<td>unit not connected</td>
</tr>
<tr>
<td>1015</td>
<td>read unexpected character</td>
</tr>
<tr>
<td>1016</td>
<td>illegal logical input field</td>
</tr>
<tr>
<td>1017</td>
<td>'new' file exists</td>
</tr>
<tr>
<td>Error</td>
<td>Message</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>1018</td>
<td>can't find 'old' file</td>
</tr>
<tr>
<td>1019</td>
<td>unknown system error</td>
</tr>
<tr>
<td>1020</td>
<td>requires seek ability</td>
</tr>
<tr>
<td>1021</td>
<td>illegal argument</td>
</tr>
<tr>
<td>1022</td>
<td>negative repeat count</td>
</tr>
<tr>
<td>1023</td>
<td>illegal operation for channel or device</td>
</tr>
<tr>
<td>1024</td>
<td>reentrant I/O</td>
</tr>
<tr>
<td>1025</td>
<td>incompatible specifiers in open</td>
</tr>
<tr>
<td>1026</td>
<td>illegal input for namelist</td>
</tr>
<tr>
<td>1027</td>
<td>error in FILEOPT parameter</td>
</tr>
<tr>
<td>1028</td>
<td>writing not allowed</td>
</tr>
<tr>
<td>1029</td>
<td>reading not allowed</td>
</tr>
<tr>
<td>1030</td>
<td>integer overflow on input</td>
</tr>
<tr>
<td>1031</td>
<td>floating-point overflow on input</td>
</tr>
<tr>
<td>1032</td>
<td>floating-point underflow on input</td>
</tr>
<tr>
<td>1051</td>
<td>default input unit closed</td>
</tr>
<tr>
<td>1052</td>
<td>default output unit closed</td>
</tr>
<tr>
<td>1053</td>
<td>direct-access READ from unconnected unit</td>
</tr>
<tr>
<td>1054</td>
<td>direct-access WRITE to unconnected unit</td>
</tr>
<tr>
<td>1055</td>
<td>unassociated internal unit</td>
</tr>
<tr>
<td>1056</td>
<td>null reference to internal unit</td>
</tr>
<tr>
<td>1057</td>
<td>empty internal file</td>
</tr>
<tr>
<td>1058</td>
<td>list-directed I/O on unformatted unit</td>
</tr>
<tr>
<td>1059</td>
<td>namelist I/O on unformatted unit</td>
</tr>
<tr>
<td>1060</td>
<td>tried to write past end of internal file</td>
</tr>
<tr>
<td>1061</td>
<td>unassociated ADVANCE specifier</td>
</tr>
<tr>
<td>1062</td>
<td>ADVANCE specifier is not 'YES' or 'NO'</td>
</tr>
<tr>
<td>1063</td>
<td>EOR specifier present for advancing input</td>
</tr>
<tr>
<td>Error</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>1064</td>
<td>SIZE specifier present for advancing input</td>
</tr>
<tr>
<td>1065</td>
<td>negative or zero record number</td>
</tr>
<tr>
<td>1066</td>
<td>record not in file</td>
</tr>
<tr>
<td>1067</td>
<td>corrupted format</td>
</tr>
<tr>
<td>1068</td>
<td>unassociated input variable</td>
</tr>
<tr>
<td>1069</td>
<td>more I/O-list items than data edit descriptors</td>
</tr>
<tr>
<td>1070</td>
<td>zero stride in subscript triplet</td>
</tr>
<tr>
<td>1071</td>
<td>zero step in implied DO-loop</td>
</tr>
<tr>
<td>1072</td>
<td>negative field width</td>
</tr>
<tr>
<td>1073</td>
<td>zero-width field</td>
</tr>
<tr>
<td>1074</td>
<td>character string edit descriptor reached on input</td>
</tr>
<tr>
<td>1075</td>
<td>Hollerith edit descriptor reached on input</td>
</tr>
<tr>
<td>1076</td>
<td>no digits found in digit string</td>
</tr>
<tr>
<td>1077</td>
<td>no digits found in exponent</td>
</tr>
<tr>
<td>1078</td>
<td>scale factor out of range</td>
</tr>
<tr>
<td>1079</td>
<td>digit equals or exceeds radix</td>
</tr>
<tr>
<td>1080</td>
<td>unexpected character in integer field</td>
</tr>
<tr>
<td>1081</td>
<td>unexpected character in real field</td>
</tr>
<tr>
<td>1082</td>
<td>unexpected character in logical field</td>
</tr>
<tr>
<td>1083</td>
<td>unexpected character in integer value</td>
</tr>
<tr>
<td>1084</td>
<td>unexpected character in real value</td>
</tr>
<tr>
<td>1085</td>
<td>unexpected character in complex value</td>
</tr>
<tr>
<td>1086</td>
<td>unexpected character in logical value</td>
</tr>
<tr>
<td>1087</td>
<td>unexpected character in character value</td>
</tr>
<tr>
<td>1088</td>
<td>unexpected character before NAMELIST group name</td>
</tr>
<tr>
<td>1089</td>
<td>NAMELIST group name does not match the name in the program</td>
</tr>
<tr>
<td>1090</td>
<td>unexpected character in NAMELIST item</td>
</tr>
<tr>
<td>1091</td>
<td>unmatched parenthesis in NAMELIST item name</td>
</tr>
<tr>
<td>Error</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>1092</td>
<td>variable not in NAMELIST group</td>
</tr>
<tr>
<td>1093</td>
<td>too many subscripts in NAMELIST object name</td>
</tr>
<tr>
<td>1094</td>
<td>not enough subscripts in NAMELIST object name</td>
</tr>
<tr>
<td>1095</td>
<td>zero stride in NAMELIST object name</td>
</tr>
<tr>
<td>1096</td>
<td>empty section subscript in NAMELIST object name</td>
</tr>
<tr>
<td>1097</td>
<td>subscript out of bounds in NAMELIST object name</td>
</tr>
<tr>
<td>1098</td>
<td>empty substring in NAMELIST object name</td>
</tr>
<tr>
<td>1099</td>
<td>substring out of range in NAMELIST object name</td>
</tr>
<tr>
<td>1100</td>
<td>unexpected component name in NAMELIST object name</td>
</tr>
<tr>
<td>1111</td>
<td>unassociated ACCESS specifier</td>
</tr>
<tr>
<td>1112</td>
<td>unassociated ACTION specifier</td>
</tr>
<tr>
<td>1113</td>
<td>unassociated BINARY specifier</td>
</tr>
<tr>
<td>1114</td>
<td>unassociated BLANK specifier</td>
</tr>
<tr>
<td>1115</td>
<td>unassociated DELIM specifier</td>
</tr>
<tr>
<td>1116</td>
<td>unassociated DIRECT specifier</td>
</tr>
<tr>
<td>1117</td>
<td>unassociated FILE specifier</td>
</tr>
<tr>
<td>1118</td>
<td>unassociated FMT specifier</td>
</tr>
<tr>
<td>1119</td>
<td>unassociated FORM specifier</td>
</tr>
<tr>
<td>1120</td>
<td>unassociated FORMATTED specifier</td>
</tr>
<tr>
<td>1121</td>
<td>unassociated NAME specifier</td>
</tr>
<tr>
<td>1122</td>
<td>unassociated PAD specifier</td>
</tr>
<tr>
<td>1123</td>
<td>unassociated POSITION specifier</td>
</tr>
<tr>
<td>1124</td>
<td>unassociated READ specifier</td>
</tr>
<tr>
<td>1125</td>
<td>unassociated READWRITE specifier</td>
</tr>
<tr>
<td>1126</td>
<td>unassociated SEQUENTIAL specifier</td>
</tr>
<tr>
<td>1127</td>
<td>unassociated STATUS specifier</td>
</tr>
<tr>
<td>1128</td>
<td>unassociated UNFORMATTED specifier</td>
</tr>
<tr>
<td>1129</td>
<td>unassociated WRITE specifier</td>
</tr>
<tr>
<td>Error</td>
<td>Message</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1130</td>
<td>zero length file name</td>
</tr>
<tr>
<td>1131</td>
<td>ACCESS specifier is not 'SEQUENTIAL' or 'DIRECT'</td>
</tr>
<tr>
<td>1132</td>
<td>ACTION specifier is not 'READ', 'WRITE' or 'READWRITE'</td>
</tr>
<tr>
<td>1133</td>
<td>BLANK specifier is not 'ZERO' or 'NULL'</td>
</tr>
<tr>
<td>1134</td>
<td>DELIM specifier is not 'APOSTROPHE', 'QUOTE', or 'NONE'</td>
</tr>
<tr>
<td>1135</td>
<td>unexpected FORM specifier</td>
</tr>
<tr>
<td>1136</td>
<td>PAD specifier is not 'YES' or 'NO'</td>
</tr>
<tr>
<td>1137</td>
<td>POSITION specifier is not 'APPEND', 'ASIS', or 'REWIND'</td>
</tr>
<tr>
<td>1138</td>
<td>RECL specifier is zero or negative</td>
</tr>
<tr>
<td>1139</td>
<td>no record length specified for direct-access file</td>
</tr>
<tr>
<td>1140</td>
<td>unexpected STATUS specifier</td>
</tr>
<tr>
<td>1141</td>
<td>status is specified and not 'OLD' for connected unit</td>
</tr>
<tr>
<td>1142</td>
<td>STATUS specifier is not 'KEEP' or 'DELETE'</td>
</tr>
<tr>
<td>1143</td>
<td>status 'KEEP' specified for a scratch file</td>
</tr>
<tr>
<td>1144</td>
<td>impossible status value</td>
</tr>
<tr>
<td>1145</td>
<td>a file name has been specified for a scratch file</td>
</tr>
<tr>
<td>1146</td>
<td>attempting to open a unit that is being read from or written to</td>
</tr>
<tr>
<td>1147</td>
<td>attempting to close a unit that is being read from or written to</td>
</tr>
<tr>
<td>1148</td>
<td>attempting to open a directory</td>
</tr>
<tr>
<td>1149</td>
<td>status is 'OLD' and the file is a dangling symbolic link</td>
</tr>
<tr>
<td>1150</td>
<td>status is 'NEW' and the file is a symbolic link</td>
</tr>
<tr>
<td>1151</td>
<td>no free scratch file names</td>
</tr>
<tr>
<td>1152</td>
<td>specifier ACCESS='STREAM' for default unit</td>
</tr>
<tr>
<td>1153</td>
<td>stream-access to default unit</td>
</tr>
<tr>
<td>1161</td>
<td>device does not support REWIND</td>
</tr>
<tr>
<td>1162</td>
<td>read permission required for BACKSPACE</td>
</tr>
<tr>
<td>1163</td>
<td>BACKSPACE on direct-access unit</td>
</tr>
<tr>
<td>1164</td>
<td>BACKSPACE on binary unit</td>
</tr>
<tr>
<td>Error</td>
<td>Message</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>1165</td>
<td>end-of-file seen while backspacing</td>
</tr>
<tr>
<td>1166</td>
<td>write permission required for ENDFILE</td>
</tr>
<tr>
<td>1167</td>
<td>ENDFILE on direct-access unit</td>
</tr>
<tr>
<td>1168</td>
<td>stream-access to sequential or direct-access unit</td>
</tr>
<tr>
<td>1169</td>
<td>stream-access to unconnected unit</td>
</tr>
<tr>
<td>1170</td>
<td>direct-access to stream-access unit</td>
</tr>
<tr>
<td>1171</td>
<td>incorrect value of POS specifier</td>
</tr>
<tr>
<td>1172</td>
<td>unassociated ASYNCHRONOUS specifier</td>
</tr>
<tr>
<td>1173</td>
<td>unassociated DECIMAL specifier</td>
</tr>
<tr>
<td>1174</td>
<td>unassociated IOMSG specifier</td>
</tr>
<tr>
<td>1175</td>
<td>unassociatedROUND specifier</td>
</tr>
<tr>
<td>1176</td>
<td>unassociatedSTREAM specifier</td>
</tr>
<tr>
<td>1177</td>
<td>ASYNCHRONOUS specifier is not 'YES' or 'NO'</td>
</tr>
<tr>
<td>1178</td>
<td>ROUND specifier is not 'UP', 'DOWN', 'ZERO', 'NEAREST', 'COMPATIBLE' or 'PROCESSOR-DEFINED'</td>
</tr>
<tr>
<td>1179</td>
<td>DECIMAL specifier is not 'POINT' or 'COMMA'</td>
</tr>
<tr>
<td>1180</td>
<td>RECL specifier is not allowed in OPEN statement for stream-access unit</td>
</tr>
<tr>
<td>1181</td>
<td>attempting to allocate an allocated array</td>
</tr>
<tr>
<td>1182</td>
<td>deallocating an unassociated pointer</td>
</tr>
<tr>
<td>1183</td>
<td>deallocating an unallocated allocatable array</td>
</tr>
<tr>
<td>1184</td>
<td>deallocating an allocatable array through a pointer</td>
</tr>
<tr>
<td>1185</td>
<td>deallocating an object not allocated by an ALLOCATE statement</td>
</tr>
<tr>
<td>1186</td>
<td>deallocating a part of an object</td>
</tr>
<tr>
<td>1187</td>
<td>deallocating a larger object than was allocated</td>
</tr>
<tr>
<td>1191</td>
<td>unallocated array passed to array intrinsic function</td>
</tr>
<tr>
<td>1192</td>
<td>illegal rank</td>
</tr>
<tr>
<td>1193</td>
<td>small source size</td>
</tr>
<tr>
<td>1194</td>
<td>zero array size</td>
</tr>
<tr>
<td>Error</td>
<td>Message</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>1195</td>
<td>negative elements in shape</td>
</tr>
<tr>
<td>1196</td>
<td>illegal kind</td>
</tr>
<tr>
<td>1197</td>
<td>nonconformable array</td>
</tr>
<tr>
<td>1213</td>
<td>asynchronous I/O on unconnected unit</td>
</tr>
<tr>
<td>1214</td>
<td>asynchronous I/O on synchronous unit</td>
</tr>
<tr>
<td>1215</td>
<td>a data edit descriptor and I/O list item type are incompatible</td>
</tr>
<tr>
<td>1216</td>
<td>current I/O list item doesn’t match with any data edit descriptor</td>
</tr>
<tr>
<td>1217</td>
<td>illegal CORR_ACCTION value</td>
</tr>
<tr>
<td>1218</td>
<td>infinite loop occurred due to I/O handler enabled</td>
</tr>
<tr>
<td>1220</td>
<td>the number of requested bytes is greater than is supported on the target platform</td>
</tr>
<tr>
<td>1221</td>
<td>data in a UNION cannot be read from or written to an incompatible file type</td>
</tr>
<tr>
<td>2001</td>
<td>invalid constant, structure, or component name</td>
</tr>
<tr>
<td>2002</td>
<td>handle not created</td>
</tr>
<tr>
<td>2003</td>
<td>character argument too short</td>
</tr>
<tr>
<td>2004</td>
<td>array argument too long or too short</td>
</tr>
<tr>
<td>2005</td>
<td>end of file, record, or directory stream</td>
</tr>
<tr>
<td>2021</td>
<td>lock not initialized (OpenMP)</td>
</tr>
<tr>
<td>2022</td>
<td>deadlock in using lock variable (OpenMP)</td>
</tr>
<tr>
<td>2023</td>
<td>lock not set (OpenMP)</td>
</tr>
</tbody>
</table>
This Appendix lists the new and changed features in this release and previous releases of the Fortran compiler.

The Sun Studio Fortran compiler, version 8.4, is a component of the Sun Studio 12 Update 1 software release.

B.1 Sun Studio 12 Update 1 Fortran Release

- Object files created by the compilers on the Solaris OS on x86 platforms or the Linux OS are incompatible with previous compiler versions if the application code contains functions with parameters or return values using _m128/_m64 data types. Users with .il inline function files, assembler code, or asm inline statements calling these functions also need to be aware of this incompatibility.
- New x86 –xtarget values woodcrest, penryn, nehalem.
- New SPARC –xtarget values ultraT2plus and sparc64vii.
- New x86 –xarch and –xchip values sse3, sse4_1, sse4_2, core2, penryn, nehalem, barcelona.
- New SPARC –xarch and –xchip values sparcima, sparc64vii, and ultraT2plus.
- The -xprofile=collect and -xprofile=use options provide improved support for profiling multi-threaded, dynamically linked applications.
- The -xcrossfile=1 option becomes an alias of the -xipo=1 option.
- On Solaris platforms, the -xpec[=yes|no] option generates a PEC binary that is recompilable for use with the Automatic Tuning System (ATS).
- On SPARC platforms, the -xdepend option is now implicitly enabled for optimization levels -x03 or higher, and is no longer included in the expansion of the -fast option.
- Support for OpenMP 3.0 tasking.
-xannotate[=yes|no] (SPARC platforms only) instructs the compiler to create binaries that can be transformed later by binary modification tools like binopt(1).

- Quad precision (REAL*16) is implemented on x86 platforms. REAL*16 is 128-bit IEEE floating point.

- The compiler normally creates temporary files in the /tmp directory. You can specify another directory by setting the TMPDIR environment variable.

- The behavior of the cpu_time() Fortran intrinsic routine is different between Solaris and Linux platforms.

- The Fortran 2003 IMPORT statement is implemented.

B.2 Sun Studio 12 Fortran Release

- The Fortran compiler is now available on the following Linux (x86 and x64) distributions: SuSe Linux Enterprise Server 9 with Service Pack 3 (or later), Red Hat Enterprise Linux 4, and other Linux distributions based on the 2.6 kernel (although these are not officially supported).

- Use -m64 to create 64-bit executables and shared libraries.

- New flags for -xarch replace obsolete flags.

- New values for -xtarget and -xchip provide code generation for the UltraSPARC T2 and SPARC64vi processors.

- New flag -fma=fused to enable generation of fused multiply-add instructions on processors that support them.

- New flag -xhwcprof enables compiler support for dataspace profiling.

- New flag -xinstrument to enable performance analysis by the Thread Analyzer

- -xregs=frameptr added to -fast on x86.

- Support for interval arithmetic on Solaris x86 platform with the -xarch=sse2 and -xia options.

- Explicit prefetch directives accepted on x86 platforms as well as SPARC platforms. (-xprefetch=explicit)

- Default format for debugging information has changed from the "stabs" standard to the "dwarf" standard format. (-xdebugformat=dwarf).
B.3 Sun Studio 11 Fortran Release

- **New -xmodel option:** The new -xmodel option lets you specify the kernel, small, or medium memory models on the 64-bit AMD architecture. If the size of your global and static variables exceeds two gigabytes, specify -xmodel=medium. Otherwise, use the default -xmodel=small setting. See "3.4.144 –xmodel=[small | kernel | medium]" on page 122.
- **The -xvector option extended for x86 SSE2 platforms:** The -xvector option enables automatic generation of calls to the vector library functions and/or the generation of the SIMD (Single Instruction Multiple Data) instructions. This option now offers the expanded syntax on x86 SSE2 platforms. See "3.4.175 –xvector=[no%lib, no%simd, %none]" on page 138.
- **STACKSIZE environment variable enhanced:** The syntax of the STACKSIZE environment variable has been enhanced to include a units keyword.
- **-xpagesize options available on x86 platforms:** Options -xpagesize, -xpagesize_heap, and -xpagesize_stack are now enabled on x86 platforms as well as SPARC. See "3.4.151 –xpagesize=size" on page 124.
- **New UltraSPARC T1 and UltraSPARC IV+ targets enabled:** Values for -xarch, -xchip, -xcache, and -xtarget support new UltraSPARC processors. See "3.4.171 –xtarget=t" on page 135.

B.4 Sun Studio 10 Fortran Release:

- **Compiling for AMD-64 Processors**
  This release introduces -xarch=amd64 and -xtarget=opteron for compiling applications to run on 64-bit x86 platforms.
- **File sharing between big-endian and little-endian platforms**
  The new compiler flag -xfilebyteorder provides cross-platform support of binary I/O files.
- **OpenMP available on Solaris OS x86 platforms**
  With this release of Sun Studio, the OpenMP API for shared-memory parallelism is available on Solaris x86 platforms as well as Solaris SPARC platforms. The same functionality is now enabled on both platforms.
- **OpenMP option -openmp=stubs no longer supported**
  An OpenMP "stubs" library is provided for user's convenience. To compile an OpenMP program that calls OpenMP library functions but ignores the OpenMP pragmas, compile the program with the -openmp option and link the object files with the libompstubs.a library. For example: % f95 omp_ignore.c -lompstubs
  Linking with both libompstubs.a and the OpenMP runtime library libmstk.so is unsupported and may result in unexpected behavior.
B.5 Sun Studio 9 Fortran Release:

- **Fortran 95 compiler released on x86 Solaris platforms:**
  This release of Sun Studio makes the Fortran compiler available on Solaris OS x86 platforms. Compile with `-xtarget` values `generic`, `native`, `386`, `486`, `pentium`, `pentium_pro`, `pentium3`, or `pentium4`, to generate executables on Solaris x86 platforms. The default on x86 platforms is `-xtarget=generic`.
  
  The following `f95` features are not yet implemented on x86 platforms and are only available on SPARC platforms:
  - Interval Arithmetic (compiler options `-xia` and `-xinterval`)
  - Quad (128-bit) Arithmetic (for example, REAL*16)
  - IEEE Intrinsic modules `IEEE_EXCEPTIONS`, `IEEE_ARITHMETIC`, and `IEEE_FEATURES`
  - The `sun_io_handler` module
  - Parallelization options such as `-autopar` and `-openmp`.

  The following `f95` command-line options are only available on x86 platforms and not on SPARC platforms: `-fprecision`, `-fstore`, `-nofstore`

  The following `f95` command-line options are only available on SPARC platforms and not on x86 platforms: `-xcode`, `-xmemalign`, `-xprefetch`, `-xcheck`, `-xia`, `-xinterval`, `-xipo`, `-xjobs`, `-xlang`, `-xlinkopt`, `-xpagesize`, `-xprofile ircache`, `-xreduction`, `-xvector`, `-depend`, `-openmp`, `-autopar`, `-vpara`, `-XlistMP`. Also, on x86 platforms `-fast` adds `-nofstore`.

**Improved Runtime Performance:**

Runtime performance for most applications should improve significantly with this release. For best results, compile with high optimization levels `-xO4` or `-xO5`. At these levels the compiler may now inline contained procedures, and those with assumed-shape, allocatable, or pointer arguments.

- **Fortran 2003 Command-Line Intrinsics:**
  The Fortran 2003 standard introduces three new intrinsics for processing command-line arguments and environment variables. These have been implemented in this release of the `f95` compiler. The new intrinsics are:
  - `GET_COMMAND(command, length, status)`
    Returns in command the entire command line that invoked the program.
  - `GET_COMMAND_ARGUMENT(number, value, length, status)`
    Returns a command-line argument in value.
  - `GET_ENVIRONMENT_VARIABLE(name, value, length, status, trim_name)`
    Return the value of an environment variable.
New and Changed Command-Line Options:

The following f95 command-line options are new in this release. See the Chapter 3 for details.

- **-xipo_archive**=none|readonly|writeback
  Allow crossfile optimization to include archive (.a) libraries. (SPARC only)

- **-xprefetch_auto_type**=[no%indirect_array_access]
  Generate indirect prefetches for a data arrays accessed indirectly. (SPARC only)

- **-xprofile_pathmap**=collect_prefix:use_prefix
  Set path mapping for profile data files. Use the -xprofile_pathmap option with the
  -xprofile=use option when profiling into a directory that is not the directory used
  when previously compiling with -xprofile=collect.

The following command-line option defaults have changed with this release of f95.

- The default for -xprefetch is -xprefetch=no%auto,explicit.
- The default for -xmemalign is -xmemalign=8i; when compiling with one of the
  -xarch=v9 options the default is -xmemalign=8f.
- The default for -xcode when compiling with one of the -xarch=v9 options is abs44.
  To compile with the defaults used in previous compiler releases, specify the following
  options explicitly:
  -xarch=v8 -xmemalign=4s -xprefetch=no for 32-bit compilation -xcode=abs64
  -xprefetch=no for 64-bit compilation

Default SPARC Architecture is V8PLUS:

The default SPARC architecture is no longer V7. Support for -xarch=v7 is limited in this
Sun Studio 9 release. The new default is V8PLUS (UltraSPARC). Compiling with -xarch=v7
is treated as -xarch=v8 because the Solaris 8 OS only supports -xarch=v8 or better.

To deploy on SPARC V8 systems (for example, SPARCStation 10), compile with -xarch=v8
explicitly. The provided system libraries run on SPARC V8 architectures.

To deploy on SPARC V7 systems (for example, SPARCStation 1), compile with -xarch=v7
explicitly. The provided system libraries use the SPARC V8 instruction set. For the Sun
Studio 9 release, only the Solaris 8 OS supports the SPARC V7 architecture. When a SPARC
V8 instruction is encountered, the OS interprets the instruction in software. The program
will run, but performance will be degraded.

- **OpenMP: Maximum Number of Threads Increased:**
  The maximum number of threads for OMP_NUM_THREADS and the multitasking library has
  increased from 128 to 256.
- **OpenMP: Automatic Scoping of Variables:**
This release of the Fortran compiler’s implementation of the OpenMP API for shared-memory parallel programming features automatic scoping of variables in parallel regions. See the OpenMP API User’s Guide for details. (OpenMP is only implemented on SPARC platforms for this release.)

B.6 Sun Studio 8 Fortran Release:

- **Enhanced -openmp option:**
  The `-openmp` option flag has been enhanced to facilitate debugging OpenMP programs. To use `dbx` to debug your OpenMP application, compile with `-openmp=noopt -g`
  You will then be able to use `dbx` to breakpoint within parallel regions and display contents of variables.

- **Multi-process compilation:**
  Specify `-xjobs=n` with `-xipo` and the interprocedural optimizer will invoke at most `n` code generator instances to compile the files listed on the command line. This option can greatly reduce the build time of large applications on a multi-cpu machine. See “3.4.133 –xjobs=n” on page 116.

- **Making assertions with PRAGMA ASSUME:**
  The `ASSUME` pragma is a new feature in this release of the compiler. This pragma gives hints to the compiler about conditions the programmer knows are true at some point in a procedure. This may help the compiler to do a better job optimizing the code. The programmer can also use the assertions to check the validity of the program during execution. See “2.3.1.8 The ASSUME Directives” on page 34, and “3.4.111 –xassume_control[=keywords]” on page 99.

- **More Fortran 2003 features:**
  The following features appearing in the Fortran 2003 standard have been implemented in this release of Fortran compiler. These are described in Chapter 4.
  - Exceptions and IEEE Arithmetic:
    New intrinsic modules `IEEE_ARITHMETIC`, and `IEEE_FEATURES` provide support for exceptions and IEEE arithmetic in the Fortran language. See “4.6.2 IEEE Floating-Point Exception Handling” on page 156.
  - Interoperability with C:
    The Fortran standard provides a means of referencing C language procedures and, conversely, a means of specifying that a Fortran subprogram can be referenced from a C function. It also provides a means of declaring global variables that are linked with external C variables. See “4.6.1 Interoperability with C Functions” on page 155.
  - **PROTECTED** Attribute
The Fortran compiler now accepts the Fortran 2003 `PROTECTED` attribute. `PROTECTED` imposes limitations on the usage of module entities. Objects with the `PROTECTED` attribute are only definable within the module that declares them. "4.6.4 PROTECTED Attribute" on page 156.

- **ASYNCHRONOUS I/O Specifier**
  The compiler recognizes the `ASYNCHRONOUS` specifier on I/O statements:
  
  ```fortran
  ASYNCHRONOUS=[ 'YES' | 'NO']
  ```

  See "4.6.5 Fortran 2003 Asynchronous I/O" on page 156.

Enhanced compatibility with legacy f77:

A number of new features enhance the Fortran compiler’s compatibility with legacy Fortran 77 compiler, `f77`. These include variable format expressions (VFE’s), long identifiers, `-arg=local`, and the `-vax` compiler option. See Chapter 3 and Chapter 4.

- **I/O error handlers:**
  Two new functions enable the user to specify their own error handling routine for formatted input on a logical unit. These routines are described in "4.7.1 I/O Error Handling Routines" on page 159, and in man pages and the Fortran Library Reference.

- **Unsigned integers:**
  With this release, the Fortran compiler accepts a new data type, `UNSIGNED`, as an extension to the language. See "4.5 Unsigned Integers" on page 153.

- **Set preferred stack/heap page size:**
  A new command-line option, `xpagesize`, enables the running program to set the preferred stack and heap page size at program startup. See "3.4.151 –xpagesize=size" on page 124.

- **Faster and enhanced profiling:**
  This release introduces the new command-line option `-xprofile_ircache=path`, to speed up the "use" compilation phase during profile feedback. See "3.4.161 –xprofile_ircache[=path]" on page 131. See also "3.4.162 –xprofile_pathmap=collect_prefix:use_prefix" on page 132.

- **Enhanced "known libraries":**
  The `-xknown_lib` option has been enhanced to include more routines from the Basic Linear Algebra library, BLAS. See "3.4.134 –xknown_lib=library_list" on page 117.

- **Link-time Optimization:**
  Compile and link with the new `-xlinkopt` flag to invoke a post-optimizer to apply a number of advanced performance optimizations on the generated binary object code at link time. See "3.4.140 –xlinkopt=[1|2|0]]" on page 119.

- **Initialization of local variables:**
A new extension to the `-xcheck` option flag enables special initialization of local variables. Compiling with `-xcheck=init_local` initializes local variables to a value that is likely to cause an arithmetic exception if it is used before it is assigned by the program. See "3.4.115 `-xcheck=keyword" on page 102.

### B.7 Sun ONE Studio 7, Compiler Collection (Forte Developer 7) Release:

- **Fortran 77 Functionality Absorbed Into Fortran 95 Compiler**
  
  This release of the Forte Developer software replaces the `f77` compiler with added functionality in the `f95` compiler. The `f77` command is a script that calls the `f95` compiler:

  ```
  the command:
  
  f77 options files libraries
  becomes a call to the `f95` compiler:
  
  f95 -f77=%all -ftrap=%none options files -lf77compat libraries
  ```

  See "4.12 Mixing Languages" on page 166 for details on Fortran 77 compatibilities and incompatibilities.

- **Fortran 77 Compatibility Mode:**
  
  The new `-f77` flag selects various compatibility features that enable the compiler to accept many Fortran 77 constructs and conventions that are normally incompatible with Fortran 95. See "3.4.24 `-f77=list" on page 60, and "4.12 Mixing Languages" on page 166.

- **Compiling “Dusty Deck” Programs That Employ Non-Standard Aliasing:**
  
  The `f95` compiler must assume that programs it compiles adhere to the Fortran 95 standard regarding aliasing of variables through subprogram calls, global variables, pointers, and overindexing. Many “dusty deck” legacy programs intentionally utilized aliasing techniques to get around shortcomings in early versions of the Fortran language. Use the new `-xalias` flag to advise the compiler about how far the program deviates from the standard and what kind of aliasing syndromes it should expect. In some cases the compiler generates correct code only when the proper `-xalias` suboption is specified. Programs that conform strictly to the standard will find some performance improvement by advising the compiler to be unconcerned about aliasing. See "3.4.108 `-xalias=keywords" on page 93, and the chapter on Porting in the Fortran Programming Guide.

- **Enhanced MODULE Features:**
  
  - New flag `-use=list` forces one or more implicit `USE` statements into each subprogram. See "3.4.99 `-use=list" on page 89.
  
  - New flag `-moddir=path` controls where the compiler writes compiled `MODULE` subprograms (.mod files). See "3.4.55 `-moddir=path" on page 75. A new environment variable, `MODDIR`, also controls where .mod files are written.
- The -Mpath flag will now accept directory paths, archive (.a) files, or module (.mod) files to search for MODULE subprograms. The compiler determines the type of the file by examining its contents; the actual file name extension is ignored. See “3.4.53 –Mpath” on page 74.

- When searching for modules, the compiler now looks first in the directory where module files are being written. See “4.9 Module Files” on page 163 for details.

Enhanced Global Program Analysis With -Xlist:

This release of the f95 compiler adds a number of new checks to the global program analysis provided by the -Xlist flag. The new -XlistMP suboption opens a new domain of static program analysis, verification of OpenMP parallelization directives. See “3.4.105 –Xlist[x]” on page 91, the Forte Developer OpenMP API User's Guide, and the chapter on Program Analysis and Debugging in the Fortran Programming Guide for details.

- Identifying Known Libraries With -xknown_lib=library:

A new option, -xknown_lib=library, directs the compiler to treat references to certain known libraries as intrinsics, ignoring any user-supplied versions. This enables the compiler to perform optimizations over library calls based on its special knowledge of the library. In this release, the known library names are limited to blas, for a subset of the BLAS routines in the Sun Performance Library, and intrinsics, for ignoring explicit EXTERNAL declarations for Fortran 95 standard intrinsics and any user-supplied versions of these routines. See “3.4.134 –xknown_lib=library_list” on page 117.

- Ignoring Dummy Argument Type in Interfaces:

A new directive, !$PRAGMA IGNORE_TKR {list_of_variables}, causes the compiler to ignore the type, kind, and rank for the specified dummy argument names appearing in a generic procedure interface when resolving a specific call. Using this directive greatly simplifies writing generic interfaces for wrappers that call specific library routines based on argument type, kind, and rank. See “2.3.1.2 The IGNORE_TKR Directive” on page 31 for details.

- Enhanced -C Runtime Array Checking:

In this f95 compiler release, runtime array subscript range checking with the -C option has been enhanced to include array conformance checking. A runtime error is produced when an array syntax statement is executed where the array sections are not conformable. See “3.4.7 –C” on page 53.

- Introducing Fortran 2003 Features:

Some new formatted I/O features proposed for the next Fortran standard have been implemented in this release of f95. These are the DECIMAL=, ROUND=, and IOMSG= specifiers, and they may appear in OPEN, READ, WRITE, PRINT, and INQUIRE statements. Also implemented are the DP, DC, RP, and RC edit descriptors. See “4.6.9 Fortran 2003 Formatted I/O Features” on page 158 for details.

- Rounding in Formatted I/O:
A new option flag, `-iorounding`, sets the default rounding mode for formatted I/O. The modes, processor-defined or compatible, correspond to the `ROUND=` specifier implemented as part of the Fortran 2003 features. See "3.4.46 `iorounding|[compatible|processor-defined]|` on page 72.

- **Obsolete Flags Removed:**

  The following flags have been removed from the `f95` command line:
  
  - `db` - `dbl`

  The following `f77` compiler flags have not been implemented in the `f95` compiler and are also considered obsolete:
  
  - `arg=local` - `i2` - `i4` - `misalign` - `oldldo` - `r8` - `vax` - `x` - `xvpara`
  - `xtypemap=int: mixed`

- **Checking for Stack Overflow:**

  Compiling with the new `-xcheck=stkovf` flag adds a runtime check for stack overflow conditions on entry to subprograms. If a stack overflow is detected, a `SIGSEGV` segment fault is raised. Stack overflows in multithreaded applications with large arrays allocated on the stack can cause silent data corruption in neighboring thread stacks. Compile all routines with `-xcheck=stkovf` if stack overflow is suspected. See “3.4.115 `xcheck=keyword`” on page 102.

- **New Default Thread Stack Size:**

  With this release, the default slave thread stack size has been increased to 4 Megabytes on SPARC V8 platforms, and 8 Megabytes on SPARC V9 platforms. See the discussion of stacks and stack sizes in the Parallelization chapter of the *Fortran Programming Guide* for details.

- **Enhanced Interprocedural Optimizations:**

  With `-xipo=1` the compiler does inlining across all source files. This release adds `-xipo=2` for enhanced interprocedural aliasing analysis and memory allocation and layout optimizations to improve cache performance. See "3.4.131 `xipo=[0|1|2]`” on page 113.

- **Control Prefetch Instructions With `-xprefetch_level=n`:**

  Use the new flag `-xprefetch_level=n` to control the automatic insertion of prefetch instructions with `-xprefetch=auto`. Use requires an optimization level of `-xO3` or greater and a target platform that supports prefetch (`-xarch platforms v8plus, v8plusa, v8plusb, v9, v9a, v9b, generic64, or native64`). See “3.4.159 `xprefetch_level=[1|2|3]`” on page 129.

Feature histories for releases prior to Forte Developer 7 can be found in the documentation sets for those earlier releases on the [http://docs.sun.com](http://docs.sun.com) web site.
This Appendix details legacy -xtarget option platform system names and their expansions. They appear here for reference purposes. The current values for UltraSPARC platforms are given under the -xtarget option description in Chapter 3. Some of the system platforms listed here may no longer be supported by recent releases of the Solaris operating environment or the Sun Studio compilers.

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.

For example:

-xtarget=sun4/15

means

-xarch=v8a -xchip=micor -xcache=2/16/1

<table>
<thead>
<tr>
<th>-xtarget=</th>
<th>-xarch</th>
<th>-xchip</th>
<th>-xcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs6400</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:2048/64/1</td>
</tr>
<tr>
<td>sc2000</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:2048/64/1</td>
</tr>
<tr>
<td>solb5</td>
<td>v7</td>
<td>old</td>
<td>128/32/1</td>
</tr>
<tr>
<td>solb6</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss1</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>ss10</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/20</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>-xtarget=</td>
<td>-xarch</td>
<td>-xchip</td>
<td>-xcache</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>ss10/30</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/40</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/402</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/41</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/412</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/50</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss10/51</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/512</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/514</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/61</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/612</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/71</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10/712</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss10 hs11</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10 hs12</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10 hs14</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10 hs21</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss10 hs22</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss1000</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss1plus</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>ss2</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss20</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/151</td>
<td>v8</td>
<td>hyper</td>
<td>512/64/1</td>
</tr>
<tr>
<td>ss20/152</td>
<td>v8</td>
<td>hyper</td>
<td>512/64/1</td>
</tr>
<tr>
<td>ss20/50</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss20/502</td>
<td>v8</td>
<td>super</td>
<td>16/32/4</td>
</tr>
<tr>
<td>ss20/51</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/512</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
</tbody>
</table>
TABLE C–1 Legacy -xtarget Expansions  (Continued)

<table>
<thead>
<tr>
<th>-xtarget=</th>
<th>-arch</th>
<th>-chip</th>
<th>-xcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss20/514</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/61</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/612</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/71</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/712</td>
<td>v8</td>
<td>super2</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss20/hs11</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss20/hs12</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss20/hs14</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss20/hs21</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss20/hs22</td>
<td>v8</td>
<td>hyper</td>
<td>256/64/1</td>
</tr>
<tr>
<td>ss2p</td>
<td>v7</td>
<td>powerup</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss4</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss4/110</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss4/85</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss5</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss5/110</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss5/85</td>
<td>v8a</td>
<td>micro2</td>
<td>8/16/1</td>
</tr>
<tr>
<td>ss600/120</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss600/140</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ss600/41</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/412</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/51</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/512</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/514</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/61</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>ss600/612</td>
<td>v8</td>
<td>super</td>
<td>16/32/4:1024/32/1</td>
</tr>
<tr>
<td>sselpc</td>
<td>v7</td>
<td>old</td>
<td>64/32/1</td>
</tr>
<tr>
<td>ssipc</td>
<td>v7</td>
<td>old</td>
<td>64/16/1</td>
</tr>
<tr>
<td>-xtarget=</td>
<td>-arch</td>
<td>-chip</td>
<td>-xcache</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>ssipx v7</td>
<td>old</td>
<td>64/32/1</td>
<td></td>
</tr>
<tr>
<td>sslc v8a micro</td>
<td>2/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sslt v7 old</td>
<td>64/32/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sslx v8a micro</td>
<td>2/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sslx2 v8a micro2</td>
<td>8/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssslc v7 old</td>
<td>64/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssyger v8a micro2</td>
<td>8/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/110 v7 old</td>
<td>2/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/15 v8a micro</td>
<td>2/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/150 v7 old</td>
<td>2/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/20 v7 old</td>
<td>64/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/25 v7 old</td>
<td>64/32/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/260 v7 old</td>
<td>128/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/280 v7 old</td>
<td>128/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/30 v8a micro</td>
<td>2/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/330 v7 old</td>
<td>128/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/370 v7 old</td>
<td>128/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/390 v7 old</td>
<td>128/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/40 v7 old</td>
<td>64/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/470 v7 old</td>
<td>128/32/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/490 v7 old</td>
<td>128/32/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/50 v7 old</td>
<td>64/32/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/60 v7 old</td>
<td>64/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/630 v7 old</td>
<td>64/32/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/65 v7 old</td>
<td>64/16/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/670 v7 old</td>
<td>64/32/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun4/690 v7 old</td>
<td>64/32/1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This appendix summarizes the directives recognized by f95 Fortran compiler:

- General Fortran Directives
- Sun Parallelization Directives
- Cray Parallelization Directives
- OpenMP Fortran 95 Directives, Library Routines, and Environment

## D.1 General Fortran Directives

General directives accepted by f95 are described in “2.3 Directives” on page 29.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C$PRAGMA keyword (a [, a ] ...) [ , keyword (a [, a ] ...) ] ,...</td>
<td>Declares a list of names of external functions as C language routines.</td>
</tr>
<tr>
<td>C$PRAGMA SUN keyword (a [, a ] ...) [ , keyword (a [, a ] ...) ] ,...</td>
<td></td>
</tr>
<tr>
<td>C$PRAGMA SPARC keyword (a [, a ] ...) [ , keyword (a [, a ] ...) ] ,...</td>
<td></td>
</tr>
</tbody>
</table>

Comment-indicator in column 1 may be c, C, !, or *. (We use C in these examples. f95 free-format must use !.)

<table>
<thead>
<tr>
<th>C Directive</th>
<th>C$PRAGMA C {list}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Declares a list of names of external functions as C language routines.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IGNORE_TKR Directive</th>
<th>C$PRAGMA IGNORE_TKR {name [, name] ...}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The compiler ignores the type, kind, and rank of the specified dummy argument names appearing in a generic procedure interface when resolving a specific call.</td>
</tr>
</tbody>
</table>
D.2 Special Fortran Directives

The following directives are only available with f95. See “4.8.2 FIXED and FREE Directives” on page 162 for details.
### D.3 Fortran OpenMP Directives

The Sun Studio Fortran compiler supports the OpenMP 3.0 Fortran API. The `-openmp` compiler flag enables these directives. (See “3.4.149 –xopenmp[=parallel|noopt|none]” on page 123).

See the OpenMP API User’s Guide for complete details.

### D.4 Sun Parallelization Directives

**Note** – Legacy Sun and Cray parallelization directives are now deprecated and ignored. Use of the OpenMP API for parallelization on Solaris SPARC and x86 platforms is preferred. See the OpenMP API User’s Guide for information on migrating legacy applications to OpenMP.

OpenMP parallelization is the preferred parallelization model with Fortran. Sun-style parallelization directives are described here for legacy applications.

#### TABLE D-3 Sun-Style Parallelization Directives Summary

<table>
<thead>
<tr>
<th>Format</th>
<th>Directive [optional_qualifiers] : initial line</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C$PAR [more_qualifiers] : continuation line</td>
</tr>
</tbody>
</table>

Fixed format, the directive-indicator may be C (as shown), C, *, or 1. Separate multiple qualifiers with commas. Characters beyond column 72 ignored unless `-e` compiler option specified.
### Sun-Style Parallelization Directives Summary (Continued)

<table>
<thead>
<tr>
<th>Directive</th>
</tr>
</thead>
</table>
| **TASKCOMMON** Directive | C$PAR TASKCOMMON block_name  
Declares variables in common block block_name as thread-private: private to a thread, but global within the thread. Declaring a common block TASKCOMMON requires that this directive appear after every common declaration of that block. |
| **DOALL** Directive | C$PAR DOALL [qualifiers]  
Parallelize DO loop that follows. Qualifiers are:  
PRIVATE(list) declare names on list PRIVATE  
SHARED(list) declare names on list SHARED  
MAXCPUS(n) use no more than n threads  
READONLY(list) listed variables not modified in loop  
SAVELAST save last value of all private variables  
STOREBACK(list) save last value of listed variables  
REDUCTION(list) listed variables are reduction variables  
SCHEDTYPE(type) use scheduling type: (default is STATIC)  
STATIC  
SELF(nchunk)  
FACTORING($m)  
GSS($m) |
| **DOSERIAL** Directive | C$PAR DOSERIAL  
Disables parallelization of the loop that follows. |
| **DOSERIAL*** Directive | C$PAR DOSERIAL*  
Disables parallelization of the loop nest that follows. |

---

**D.5 Cray Parallelization Directives**

**Note** – Legacy Sun and Cray parallelization directives are now deprecated and ignored. Use of the OpenMP API for parallelization on Solaris SPARC and x86 platforms is preferred. See the OpenMP API User’s Guide for information on migrating legacy applications to OpenMP.

Cray-style parallelization directives are detailed in the chapter on parallelization in the *Fortran Programming Guide*. Requires `-mp=cray` compiler option.
<table>
<thead>
<tr>
<th><strong>TABLE D–4</strong> Cray Parallelization Directives Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format</strong></td>
</tr>
<tr>
<td>CMIC$ \text{directive qualifications} : initial line</td>
</tr>
<tr>
<td>CMIC$[\text{more qualifiers}] : continuation line</td>
</tr>
<tr>
<td>Fixed format. Directive-indicator may be C (as shown here), c, *, or !. With f95 free-format, leading blanks can appear before !MIC$.</td>
</tr>
</tbody>
</table>

| **DOALL Directive**                          |
| CMIC$ \text{DOALL} \text{ SHARED(list), PRIVATE(list) }[, \text{more qualifiers}] |
| Parallelize loop that follows. Qualifiers are: |
| Scoping qualifiers are required (unless list is empty)—all variables in the loop must appear in a \text{PRIVATE} or \text{SHARED} clause: |
| \text{PRIVATE(list) declare names on list PRIVATE} |
| \text{SHARED(list) declare names on list SHARED} |
| \text{AUTOSCOPE automatically determine scope of variables} |
| The following are optional: |
| MAXCPUS(n) use no more than n threads |
| SAVELAST save last value of all private variables. Only one scheduling qualifier may appear: |
| GUIDED equivalent to Sun-style GSS(64) |
| SINGLE equivalent to Sun-style SELF(1) |
| CHUNKSIZE(n) equivalent to Sun-style SELF(n) |
| NUMCHUNKS(m) equivalent to Sun-style SELF(n/m) |
| The default scheduling is Sun-style STATIC, for which there is no Cray-style equivalent. Interpretations of these scheduling qualifiers differ between Sun and Cray style. Check the Fortran Programming Guide for details. |

| **TASKCOMMON Directive**                        |
| CMIC$ \text{TASKCOMMON block_name} |
| Declares variables in the named common block as thread-private—private to a thread, but global within the thread. Declaring a common block TASKCOMMON requires that this directive appear immediately after every common declaration of that block. |

| **DO SERIAL Directive**                          |
| CMIC$ \text{DO SERIAL} |
| Disables parallelization of the loop that follows. |

| **DO SERIAL* Directive**                          |
| CMIC$ \text{DO SERIAL*} |
| Disables parallelization of the loop nest that follows. |
Index

Numbers and Symbols

!DIR$ in directives, 161
#define, 27
#include, 27

A
abrupt_underflow, 64
accessible documentation, 15-16
address space, 92
aliasing, 93
-xalias, 93
align
See also data
-dalign, 55
data in COMMON with -aligncommon, 51
alignment of data types, 147
ALLOCATABLE, extensions, 157
analyzer compile option, xf, 107
application registers (SPARC), 133
arguments, agreement, Xlist, 91
arithmetic, See floating-point
array bounds checking, 53
asa, Fortran print utility, 18
assembly code, 85
ASSUME directive, 34
auto-read (dbx), 134

B
backward compatibility, options, 49
binary I/O, 159-161
binding, dynamic/shared libraries, 57
Boolean
constant, alternate forms, 144
type, constants, 143

C
c(…) directive, 31
cache
padding for, 80
specify hardware cache, 101
CALL, inlining subprogram calls with -inline, 71
case, preserve upper and lower case, 88
CDIR$ in directives, 161
code size, 134
command-line
help, 21-22
unrecognized options, 28
comments, as directives, 161
COMMON
alignment, 51
global consistency, -Xlist, 91
padding, 80
TASKCOMMON consistency checking, 105
compatibility
Fortran 77, 60, 167
forward, 166
with C, 166
Index

compile and link, 25, 27
   and -B, 53
   build a dynamic shared library, 69
compile only, 54
dynamic (shared) libraries, 57
compiler
   command line, 25
   driver, show commands with -dryrun, 57
options summary, 43-50
show version, 89
timing, 88
verbose messages, 89
constant arguments, -copyargs, 54
continuation lines, 58, 141
conventions, file name suffixes, 25
cpp, C preprocessor, 27, 55, 59
Cray
   pointer, 149
   pointer and Fortran pointer, 150
cross reference table, Xlist, 91

data
dependence, -depend, 56
data
   alignment with -dbl_align_all, 56
   alignment with -f, 59
   alignment with -xmemalign, 120-121
   COMMON, alignment with aligncommon, 51
   mappings with -xtypemap, 137
   promote constants to REAL*8, 84
   size and alignment, 147
dbx, compile with -g option, 69
documentation
   accessing, 15-16
documentation index, 15
dynamic library
   build, -G, 69
   name a dynamic library, 70
default
   data sizes and alignment, 147
   include file paths, 71
   define symbol for cpp, Dname, 54
directives
   ASSUME, 34-36
   FIXED, 162-163
   Fortran 77, 29
   FREE, 162-163
   IGNORE_TKR, 31
   loop unrolling, 32
   OpenMP (Fortran), 36, 199
   optimization level, 33
   parallelization, 36, 163
   special Fortran, 161
   summary of all directives, 197
   weak linking, 33
directory, temporary files, 87-88
documentation
   accessing, 15-16
documentation index, 15
dynamic library
   build, -G, 69
   name a dynamic library, 70
documentation
   accessing, 15-16
documentation index, 15
dynamic library
   build, -G, 69
   name a dynamic library, 70
documentation
   accessing, 15-16
documentation index, 15
dynamic library
   build, -G, 69
   name a dynamic library, 70
documentation
   accessing, 15-16
documentation index, 15
dynamic library
   build, -G, 69
   name a dynamic library, 70
documentation
   accessing, 15-16
documentation index, 15
extensions (Continued)
  non-ANSI, -ansi flag, 51
  other I/O, 159-161
  stream I/O, 158
VALUE, 157
  VAX structures and unions, 152
external C functions, 31
external names, 59

F
f95 command line, 25, 41
fdumpmod for viewing module contents, 29, 165
features and extensions, 18
features
  Fortran 95, 141
  release history, 183
FFLAGS environment variable, 38
file names
  recognized by the compiler, 25, 142
file
  executable, 25
  object, 25
  size too big, 38
FIXED directive, 162-163
fixed-format source, 63
flags, See options
floating-point
  interval arithmetic, 112-113
  non-standard, 64
  preferences, -f simple, 67
  rounding, 66
  trapping mode, 68
FLUSH statement, 159
form, tab, 141
Fortran 2003, 155
Fortran 95
  case, 143
  features, 141
  Forte Developer 7 release, 190-192
  I/O extensions, 159-161
  linking with Fortran 77, 173-174
Fortran
  compatibility with legacy, 51, 60, 167

Fortran (Continued)
  directives, 161-163
  features and extensions, 18
  handling nonstandard Fortran 77 aliasing, 174
  incompatibilities with legacy, 172
  modules, 163-165
  preprocessor, 55
    invoking with -F, 59
    utilities, 18-19
fpp, Fortran preprocessor, 27, 55, 59, 65
FREE directive, 162-163
free-format source, 66
fsecond-underscore, 59
fsplit, Fortran utility, 19
function-level reordering, 107
function, external C, 31

G
global program checking, -Xlist, 91
global symbols, weak, 33
gprof, -pg, profile by procedure, 82

H
hardware architecture, 95, 103
heap page size, 124, 125
help
  command-line, 21-22
  README information, 110
hexadecimal, 145
Hollerith, 145

I
I/O extensions, 159-161
IGNORE_TKR directive, 31
IMPORT statement, 159
INCLUDE files, 70
  floatingpoint.h, 174
  system.inc, 36
incompatibilities, Fortran 77, 172
Index

initialization of local variables, 102
inline
  templates, -libmil, 73
  with -fast, 62
inlining
  automatic with -04, 79
  with -inline, 71
installation, path, 71
interfaces, library, 36
interval arithmetic
  -xia option, 111-112
  -xinterval option, 112-113
intrinsics
  extensions, 165
  interfaces, 36
  legacy Fortran, 174
invalid, floating-point, 68
ISA, instruction set architecture, 95

linking
  consistent compile and link, 27
  consistent with compilation, 27-28
  disable system libraries, 77
  enable dynamic linking, shared libraries, 57
  linker -Mmapfile option, 107
  separate from compilation, 27
  specifying libraries with -l, 73
  weak names, 33
  with automatic parallelization, -autopar, 52
  with compilation, 25
list of directives, 197
list of options, 70
loop
  automatic parallelization, 52
  dependence analysis, -depend, 56
  executed once, -onetrip, 80
  parallelization messages, 74
  unrolling with -unroll, 89
  unrolling with directive, 32

L
large files, 38
legacy compiler options, 49
libm, searched by default, 73
libraries, Sun Performance Library, 19
library
  build, -G, 69
  disable system libraries, 77
  dynamic search path in executable, 83
  interfaces, 36
  linking with -l, 73
  multithread-save, 76
  name a shared library, 70
  path to shared library in executable, 78
  position-independent and pure, 139
  Sun Performance Library, 118
limit
  command, 39
limits, Fortran compiler, 143
limit
  stack size, 86
linear algebra routines, 118
link-time optimizations, 119

M
macro options, 48-49
man pages, 19
math library
  and -L dir option, 73
  optimized version, 118
memory
  actual real memory, display, 39
  limit virtual memory, 39
  optimizer out of memory, 38
messages
  parallelization, 74,90
  runtime, 175
  suppress with -silent, 85
  verbose, 89
misaligned data, specifying behavior, 120
.mod file, module file, 163
MODDIR environment variable, 76
modules, 163-165
  creating and using, 28
  default path, 76
fdumpmod, 29
modules (Continued)

fdumpmod for displaying module files, 165
.mod file, 163
-use, 164
multithread-safe libraries, 76
multithreading, See parallelization

N

name
argument, do not append underscore, 31
object, executable file, 80
nonstandard_arithmetic(), 64
numeric sequence type, 51

O

object files
compile only, 54
name, 80
object library search directories, 72
obsolete options, 49-50
octal, 144
one-trip DO loops, 80
OpenMP, 36
directives summary, 199

OPT directive, 33
-xmaxopt option, 120
optimization (Continued)
specify instruction set architecture, 95
specify processor, 103
target hardware, 76
vector library transformations with
-xvector, 138-139
with -fast, 61
with debugging, 69

OPTIONS environment variable, 38
options
-a, 50
-a (obsolete), 49
-aligncommon, 51
-ansi, 51
-arg=local, 51
-autopar, parallelize automatically, 52
-Bdynamic, 52
-Bstatic, 52
-c, check subscripts, 53-54
-c, compile only, 54
-cg89, -cg92 (obsolete), 49
commonly used, 48
-copyargs, allow stores to literal arguments, 54
-Dname, define symbol, 54
-dalign, 55-56, 62
dbl_align_all, force data alignment, 56
depend, 62
data dependency analysis, 56-57
dn, 57
dryrun, 57
dy, 57
-e, extended source lines, 57-58
-erroff, suppress warnings, 58
-errtags, display message tag with warnings, 58
-errwarn, error warnings, 58
-explicitpar (obsolete), 50
-ext_names, externals without underscore, 59
-F, 59
-f, align on 8-byte boundaries, 59-60
-f77, 60
-fast, 61-63
-fixed, 63
-flags, 63
-fma, 63
options (Continued)

-fnonstd, 64
-fns, 62, 64-65
-fpp, Fortran preprocessor, 65
-fprecision, x86 precision mode, 66
-free, 66
-fround=r, 66
-fsimple, 62
  simple floating-point model, 67-68
-fstore, 68
-ftrap, 68-69
-G, 69
-g, 69
  grouped by function, 43
  -hname, 70
  -help, 70
  -Idir, 70-71
  -i8 — use -xtypemap=integer:64 instead, 71
  -inline, 71-72
  -iorounding, 72
-Kpic, 72
-KPIC, 72
-Ldir, 72-73
-llibrary, 73
legacy, 49

-libmil, 62, 73
-loopinfo, show parallelization, 74
-Mdir, f95 modules, 163
-m32 | -m64, 75
macros, 48-49

-maddir, 75
-mp (obsolete), 50
-nt, multithread safe libraries, 76

-native, 76

-native (obsolete), 50

-noautocupar, 76
-nodepend, 76

-nofstore, 77

-nolib, 77

-nolibmil, 77

-noqueue (obsolete), 50

-noreduction, 77

-norunpath, 77-78

-On, 62, 78

options (Continued)

-o, output file, 80
obsolete, 49-50
obsolete f77 flags not supported, 173

-onetrip, 80

-openmp, 80
order of processing, 42
-p (obsolete), 50
-p, profile by procedure, 80
-pad=p, 62, 80-81
-parallel (obsolete), 50
pass option to compilation phase, 83

-pg, profile by procedure, 82
-pic, 82

-PIC, 83

-pic (obsolete), 50

-PIC (obsolete), 50

-Qoption, 83

-Rlist, 83-84

-r8const, 84
-recl=a[,b], 84

Reference to all option flags, 50

-s, 85

-s, 85

-sb, -sbfast (obsolete), 50

-sb, obsolete, 85

-sbfast, 85

-silent, 85

-stackvar, 86-87, 132

-stop_status, 87
summary, 43-50

syntax on command line, 41

-temp, 87-88

-time, 88

-U, do not convert to lowercase, 88

-uname, undefine preprocessor macro, 88

-u, 88

unrecognized, 28

-unroll, unroll loops, 89

-use, 164-165

-V, 89

-v, 89

-vax, 89-90

-vpara, 90-91
options (Continued)

-w, 91
-xa, 92
-xaddr32, 92
-xalias=list, 93-95
-xannotate[=yes|no], 95
-xarch=isa, 95-99
-xassume_control, 35, 99
-xautopar, 100
-xbinopt, 100-101
-xcache=c, 101-102
-xchip=c, 103-104
-xcode=c, 104
-xcommoncheck, 105-106
-xcrossfile, 106
-xdepend, 107
-xF, 107
-xhasc, Hollerith as character, 110
-xhelp=h, 110
-xhwprof, 111
-xia, interval arithmetic, 111-112
-xinline, 112
-xinstrument, 112
-xinterval=v for interval arithmetic, 112-113
-xipo, interprocedural optimizations, 113-115
-xipo_archive, 115
-xjobs, multiprocessor compilation, 116
-xknown_lib, optimize library calls, 117
-xlang=f77, link with Fortran 77 libraries, 117-118
-xlbind, 118
-xlibmopt, 62, 118
-xlic_lib=sunperf, 118
-xlicinfo (obsolete), 118
-xlinkopt, 119-120
-xlinkopt, link-time optimizations, 119-120
-xlist, global program checking, 91-92
-xloopinfo, 120
-xmaxopt, 120
-xmemalign, 120-121
-xnolib, 122
-xnolibmopt, 122
-xO, 123
-xopenmp, 123-124

options (Continued)

-xpagesize, 124-125
-xpagesize_heap, 125
-xpagesize_stack, 125
-xpec, 125
-xpg, 126
-xpp=p, 126
-xprefetch, 34
-xprefetch_auto_type, 128
-xprofile ircache, 131
-xprofile=p, 129-131
-xprofile_pathmap=param, 132
-xrecursive, 132
-xreduction, 133
-xregs=r, 133
-xs, 134
-xsafe=mem, 134
-xsb, 134
-xsbfast, 134
-xspace, 134
-xtarget=native, 62
-xtarget=t, 135-137, 193
-xt ime, 137
-xtypemap, 137-138
-xunroll, 138
-xvector, 62, 138-139
-ztext, 139

order of processing, options, 42
order of, functions, 107
overflow
stack, 86
trap on floating-point, 68
overindexing, aliasing, 93

P

padding, 80
page size, setting stack or heap, 124, 125
parallelization
automatic, 52
directives, 163
directives (f77), 36
loop information, 74
parallelization (Continued)
  messages, 90
  OpenMP, 36, 123-124
  OpenMP directives summarized, 199
  reduction operations, 84
  with multithreaded libraries, 76
parameters, global consistency, Xlist, 91
passes of the compiler, 89
path
  #include, 70
  dynamic libraries in executable, 83
  library search, 72
  to standard include files, 71
Pentium, 137
performance library, 118
performance
  optimization, 61
  Sun Performance Library, 19
PIPELOOP directive, 33-34
pointer, 149
pointer
  aliasing, 93
position-independent code, 82, 83, 104
POSIX library, not supported, 173
pragma, See directives
precision on x86
  -fp, 66
  -fstore, 68
PREFETCH directive, 34
preprocessor, source file
  define symbol, 54
  force -fpp, 65
  fpp.cpp, 27
  specify with -xpp=p, 126
  undefine symbol, 88
preserve case, 88
print, asa, 18
processor, specify target processor, 103
prof, p, 80
profile data path map, 132
profiling
  -pg, -gprof, 82
  -xprofile, 129
R
  range of subscripts, 53
README file, 20-21, 110
recursive subprograms, 132
register usage, 133
release history, 183
reorder functions, 107
rounding, 66, 67
S
search, object library directories, 72
set, #include path, 70
shared library
  build, -G, 69
  disallow linking, -dn, 57
  name a shared library, 70
  pure, no relocations, 139
shell, limits, 39
SIGFPE, floating-point exception, 64
size of compiled code, 134
source file, preprocessing, 27
source format
  mixing format of source lines (f95), 143
  options (f95), 142
source lines
  extended, 58
  fixed-format, 63
  free-format, 66
  line length, 141
  preprocessor, 126
  preserve case, 88
SPARC platform
  cache, 101
  chip, 103
  code address space, 104
  instruction set architecture, 96
  register usage, -xregs, 133
  -xtarget expansions, 193
stack overflow, 102
stack
  increase stack size, 86
  overflow, 86
  setting page size, 124, 125
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STACKSIZE</strong> environment variable</td>
<td>87</td>
</tr>
<tr>
<td>standard numeric sequence type</td>
<td>51</td>
</tr>
<tr>
<td>standard, include files</td>
<td>71</td>
</tr>
<tr>
<td>standards</td>
<td></td>
</tr>
<tr>
<td>conformance</td>
<td>17</td>
</tr>
<tr>
<td>identify non-ANSI extensions, <strong>ansi</strong> flag</td>
<td>51</td>
</tr>
<tr>
<td>static, binding</td>
<td>57</td>
</tr>
<tr>
<td><strong>STOP</strong> statement, return status</td>
<td>87</td>
</tr>
<tr>
<td>stream I/O</td>
<td>158</td>
</tr>
<tr>
<td><strong>strict</strong> (interval arithmetic)</td>
<td>113</td>
</tr>
<tr>
<td>strip executable of symbol table, <strong>s</strong></td>
<td>85</td>
</tr>
<tr>
<td>suffix</td>
<td></td>
</tr>
<tr>
<td>of file names recognized by compiler</td>
<td>25</td>
</tr>
<tr>
<td>of file names recognized by compiler (f95)</td>
<td>142</td>
</tr>
<tr>
<td>suppress</td>
<td></td>
</tr>
<tr>
<td>implicit typing</td>
<td>88</td>
</tr>
<tr>
<td>linking</td>
<td>54</td>
</tr>
<tr>
<td>warnings</td>
<td>91</td>
</tr>
<tr>
<td>warnings by tag name, <strong>erroff</strong></td>
<td>58</td>
</tr>
<tr>
<td><strong>swap</strong> command</td>
<td>39</td>
</tr>
<tr>
<td>swap space</td>
<td></td>
</tr>
<tr>
<td>display actual swap space</td>
<td>39</td>
</tr>
<tr>
<td>limit amount of disk swap space</td>
<td>38</td>
</tr>
<tr>
<td>symbol table, for <strong>dbx</strong></td>
<td>69</td>
</tr>
<tr>
<td>syntax</td>
<td></td>
</tr>
<tr>
<td>compiler command line</td>
<td>41</td>
</tr>
<tr>
<td><strong>f95</strong> command</td>
<td>25, 41</td>
</tr>
<tr>
<td>options on compiler command line</td>
<td>41</td>
</tr>
<tr>
<td><strong>system.inc</strong>, 36-37</td>
<td></td>
</tr>
</tbody>
</table>

**U**

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ulimit</strong> command</td>
<td>39</td>
</tr>
<tr>
<td>underflow</td>
<td></td>
</tr>
<tr>
<td>gradual</td>
<td>65</td>
</tr>
<tr>
<td>trap on floating-point</td>
<td>68</td>
</tr>
<tr>
<td>underscore</td>
<td>59</td>
</tr>
<tr>
<td>do not append to external names</td>
<td>31</td>
</tr>
<tr>
<td>unrecognized options</td>
<td>28</td>
</tr>
<tr>
<td><strong>UNROLL</strong> directive</td>
<td>32</td>
</tr>
<tr>
<td>usage, compiler</td>
<td>25</td>
</tr>
<tr>
<td>utilities</td>
<td>18-19</td>
</tr>
</tbody>
</table>

**V**

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables</td>
<td></td>
</tr>
<tr>
<td>alignment</td>
<td>147</td>
</tr>
<tr>
<td>local</td>
<td>86</td>
</tr>
<tr>
<td>undeclared</td>
<td>88</td>
</tr>
<tr>
<td>VAX VMS Fortran extensions</td>
<td>89, 152</td>
</tr>
<tr>
<td>version, id of each compiler pass</td>
<td>89</td>
</tr>
</tbody>
</table>

**W**

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>warnings</td>
<td></td>
</tr>
<tr>
<td>message tags</td>
<td>58</td>
</tr>
<tr>
<td>suppress messages</td>
<td>91</td>
</tr>
<tr>
<td>suppress with <strong>erroff</strong></td>
<td>58</td>
</tr>
<tr>
<td>undeclared variables</td>
<td>88</td>
</tr>
<tr>
<td>use of non-standard extensions</td>
<td>51</td>
</tr>
<tr>
<td><strong>WEAK</strong> directive</td>
<td>33</td>
</tr>
<tr>
<td>weak linker symbols</td>
<td>33</td>
</tr>
<tr>
<td><strong>widestneed</strong> (interval arithmetic)</td>
<td>113</td>
</tr>
</tbody>
</table>

**T**

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab, form source tab</td>
<td>141</td>
</tr>
<tr>
<td>tape I/O, not supported</td>
<td>173</td>
</tr>
<tr>
<td><strong>tcov</strong>, new style with <strong>xprofile</strong></td>
<td>130</td>
</tr>
<tr>
<td>templates, inline</td>
<td>73</td>
</tr>
<tr>
<td>temporary files, directory for, 87-88</td>
<td></td>
</tr>
<tr>
<td>trapping</td>
<td></td>
</tr>
<tr>
<td>floating-point exceptions</td>
<td>68</td>
</tr>
<tr>
<td>on memory</td>
<td>134</td>
</tr>
<tr>
<td>type declaration alternate form</td>
<td>146</td>
</tr>
</tbody>
</table>