Using the x86 Open64 Compiler Suite

For x86 Open64 version 4.2.3

Advanced Micro Devices, Inc. & The Open64 Developer Community

Published by:

Advanced Micro Devices, Inc.
One AMD Place
Sunnyvale, CA 94088-3453 USA

Website: developer.amd.com
Tel 408-749-4000

Short Contents

Table of Contents

Introduction

This manual documents how to use the x86 Open64 compilers, as well as their features and incompatibilities, and how to report bugs. It corresponds to x86 Open64 version 4.2.3.

This manual does *not* include the internals of the Open64 compiler. The x86 Open64 Compiler Suite lets you build and optimize C, C++, and Fortran applications for the Linux® OS (operating system). The x86 Open64 Compiler Suite is designed to be used on the command line. Through out this user guide the x86 Open64 Compiler Suite may be referred to as "Open64" or "Open64 compilers".

1 The x86 Open64 Compilers

The x86 Open64 compiler system is designed to generate code for x86, AMD64 (AMD x86-64 Architecture), and Intel64 (Intel x86-64 Architecture) applications. The x86 Open64 environment provides the developer the necessary options when building and optimizing C, C++, and Fortran applications targeting 32-bit and 64-bit Linux® platforms.

The x86 Open64 compiler system offers many advanced optimizations including global optimization, vectorization, interprocedureal analysis, feedback directed optimizations and loop transformations. The x86 Open64 compiler system also provides micro-architecture-specific code generation.

IA-32 applications (32-bit) can run on all x86, AMD64, and Intel64 based Linux systems. x86-64 applications (64-bit) can only run on AMD64 or Intel64 based Linux systems.

For more information about the compiler features and other components, see the Release Notes.

1.1 Programming Languages Supported

Open64 is an integrated distribution of compilers for three major programming languages: C, C++, and Fortran. Open64 has been retargeted to a number of x86 architectures. The language-independent component of Open64 includes the majority of the optimizers, as well as the 'code generators' that generate machine code for various processors (e.g. AMD64, IA-32, Intel64).

The part of a compiler that is specific to a particular language is called the 'front-end'. In addition to the front-ends that are integrated components of Open64, there are several other front ends that are maintained separately e.g., the Fortran front-end.

The C preprocessor is an integral feature of the C/C++ programming languages.

This documentation assumes that you are familiar with the C, C++, and Fortran programming languages and with your processor's architecture. You should also be familiar with the host computer's operating system.

Each compiler is invoked using its own compiler driver. The C compiler is invoked using `opencc`, the C++ compiler is invoked using `openCC`, and Fortran is invoked using `openf90`. When we talk about compiling one of those languages, we might refer to that compiler by its own name, or as Open64. Either is correct.

Historically, compilers for many languages, including C++ and Fortran, have been implemented as "preprocessors" which emit another high level language such as C. None of the compilers included in Open64 are implemented this way; they all generate machine code directly. This sort of preprocessor should not be confused with the *C preprocessor*, which is an integral feature of the C, C++, and Fortran languages.

1.2 Language Standards Supported

For each language compiled by Open64 for which there is a standard, Open64 attempts to follow one or more versions of that standard, possibly with some exceptions, and possibly with some extensions.

Open64 supports three versions of the C standard, although support for the most recent version is not yet complete.

The original ANSI C standard (X3.159-1989) was ratified in 1989 and published in 1990. This standard was ratified as an ISO standard (ISO/IEC 9899:1990) later in 1990. There were no technical differences between these publications, although the sections of the ANSI standard were renumbered and became clauses in the ISO standard. This standard, in both its forms, is commonly known as *C89*, or occasionally as *C90*, from the dates of ratification. The ANSI standard, but not the ISO standard, also came with a Rationale document. To select this standard in Open64, use one of the options '`-ansi`', '`-std=c89`' or '`-std=iso9899:1990`'; to obtain all the diagnostics required by the standard, you should also specify '`-pedantic`' (or '`-pedantic-errors`' if you want them to be errors rather than warnings). See Section 3.5 [Options Controlling C Dialect], page 36.

Errors in the 1990 ISO C standard were corrected in two Technical Corrigenda published in 1994 and 1996. Open64 does not support the uncorrected version.

An amendment to the 1990 standard was published in 1995. This amendment added digraphs and `__STDC_VERSION__` to the language, but otherwise concerned the library. This amendment is commonly known as *AMD1*; the amended standard is sometimes known as *C94* or *C95*. To select this standard in Open64, use the option '`-std=iso9899:199409`' (with, as for other standard versions, '`-pedantic`' to receive all required diagnostics).

A new edition of the ISO C standard was published in 1999 as ISO/IEC 9899:1999, and is commonly known as *C99*. Open64 has incomplete support for this standard version; see `http://gcc.gnu.org/gcc-4.2/c99status.html` for details. To select this standard, use '`-std=c99`' or '`-std=iso9899:1999`'. (While in development, drafts of this standard version were referred to as *C9X*.)

Errors in the 1999 ISO C standard were corrected in two Technical Corrigenda published in 2001 and 2004. Open64 does not support the uncorrected version.

For references to Technical Corrigenda, Rationale documents and information concerning the history of C that is available online, see `http://gcc.gnu.org/readings.html`

By default, Open64 provides some extensions to the C language that on rare occasions conflict with the C standard. Use of the '`-std`' options listed above will disable these extensions where they conflict with the C standard version selected. You may also select an extended version of the C language explicitly with '`-std=gnu89`' (for C89 with GNU extensions) or '`-std=gnu99`' (for C99 with GNU extensions). The default, if no C language dialect options are given, is '`-std=gnu89`'; this will change to '`-std=gnu99`' in some future release when the C99 support is complete. Some features that are part of the C99 standard are accepted as extensions in C89 mode.

The Open64 C++ compiler conforms to ISO/IEC 14882: 1998(E), Programming Languages-C++ standard.

The Open64 C/C++ compiler generates code which complies with:

- the C/C++ Application Binary Interface (ABI) as defined by GCC,
- and with the x86-32 ABI and x86-64 ABI.

The compiler supports most of the generally used command-line options supported by GCC and a number of the GNU extensions.

The Open64 Fortran compiler complies with the ISO/IEC 1539-1:1997 (Fortran 95) standards. The compiler is also able to compile essentially all standard-compliant Fortran 90 and Fortran 77 programs. It also supports the ISO/IEC TR-15581

enhancements to allocatable arrays, and the OpenMP Application Program Interface v2.5 (`http://www.openmp.org/drupal/mp-documents/spec25.pdf`) specification. The Open64 Fortran compiler conforms to:

- ISO/IEC TR 15580: Fortran floating point exception handling.
- ISO/IEC TR 15581: Fortran enhanced data type facilities.
- ISO/IEC 1539-2: Varying length character strings.
- ISO/IEC 1539-3: Conditional compilation.
- ISO/IEC 1539:1991 (Fortran 90) Programming languages-Fortran.

The Open64 Fortran compiler also: supports legacy FORTRAN 77 (i.e. ANSI X3.9-1978) programs, ABI compatible with GNU FORTRAN 77 programs, and generates code which complies with the x86-32 ABI and x86-64 ABI.

Although Open64 Fortran focuses on implementing the Fortran 95 standard for the time being, a few Fortran 2003 features are currently available. These include partial conformance with ISO/IEC 1539-1:2004 (Fortran 2003) Programming Languages-Fortran.

1.3 How To Get Help

AMD Technical Support is available to x86 Open64 AMD platform developers, for more information please visit `http://developer.amd.com/cpu/open64/`. The x86 Open64 compiler can be downloaded and supported from the AMD Developer Central Web Site.

Available to the developer, at the x86 Open64 website, is x86 Open64 compiler information, including:

- Comprehensive product documentation
- Technical user forums
- User friendly knowledge-base
- AMD 'Service Request' help desk

AMD provides support of the x86 Open64 Compiler Suite distributed by AMD on a limited bases. AMD's product offering is comprised of x86 Open64 compilers and libraries. This applies to current and past x86 Open64 releases distributed by AMD, depending on release life cycle. If you think you have found a bug in x86 Open64, please report it following the instructions in Section 1.4.2 [Bug Reporting], page 6.

AMD Technical Support provides services to customers using the x86 Open64 compiler to develop or port applications to AMD platforms within an enterprise Linux environment. AMD offers specialized support to the developer throughout the development and test cycle of their application. For further information on AMD Technical Support, see the x86 Open64 Technical Support Guide and Service Level Agreement at `http://developer.amd.com/cpu/open64/`.

For detailed information on host system requirements, see the x86 Open64 Installation Guide and/or the Release Notes.

1.4 Reporting Bugs

Your bug reports play an essential role in making x86 Open64 reliable.

When you encounter a problem, the first thing to do is to see if it is already known. If it isn't known, then you should report the problem.

1.4.1 Have You Found a Bug?

If you are not sure whether you have found a bug, here are some guidelines:

- If the compiler gets a fatal signal, for any input whatever, that is a compiler bug. Reliable compilers never crash.
- If the compiler produces invalid assembly code, for any input whatever (except an `asm` statement), that is a compiler bug, unless the compiler reports errors (not just warnings) which would ordinarily prevent the assembler from being run.
- If the compiler produces valid assembly code that does not correctly execute the input source code, that is a compiler bug.

  However, you must double-check to make sure, because you may have a program whose behavior is undefined, which happened by chance to give the desired results with another C/C++ or Fortran compiler.

  For example, in many nonoptimizing compilers, you can write '`x;`' at the end of a function instead of '`return x;`', with the same results. But the value of the function is undefined if `return` is omitted; it is not a bug when Open64 produces different results.

  Problems often result from expressions with two increment operators, as in `f (*p++, *p++)`. Your previous compiler might have interpreted that expression the way you intended; Open64 might interpret it another way. Neither compiler is wrong. The bug is in your code.

  After you have localized the error to a single source line, it should be easy to check for these things. If your program is correct and well defined, you have found a compiler bug.
- If the compiler produces an error message for valid input, that is a compiler bug.
- If the compiler does not produce an error message for invalid input, that is a compiler bug. However, you should note that your idea of "invalid input" might be someone else's idea of "an extension" or "support for traditional practice".
- If you are an experienced user of one of the languages x86 Open64 supports, your suggestions for improvement of Open64 are welcome in any case.

1.4.2 How and where to Report Bugs

Bugs should be reported to the x86 Open64 bug database. Please refer to

`http://developer.amd.com/cpu/open64`

for up-to-date instructions regarding how to submit bug reports. Copies of this file in HTML ('`bugs.html`') and plain text ('`BUGS`') are also part of x86 Open64 releases.

1.5 Contributing to Open64 Development

Supported binary releases and source snapshots of Open64 are available at:

`http://developer.amd.com/cpu/open64.`

These sources are periodically merged into the main SVN tree at:

`http://svn.open64.net.`

If you would like to work on improvements to Open64, please read the advice at these URLs:

`http://www.open64.net/home.html`
`http://www.open64.net/about-open64.html`

for information on how to make useful contributions and avoid duplication of effort. Suggested projects are listed at:

`http://wiki.open64.net/index.php/Main_Page`.

2 Using the x86 Open64 Compiler

The x86 Open64 Compiler Suite uses the GCC front-ends to handle programs written in C and C++. Programs written in Fortran use the SGI/Cray front-end. The C/C++ and Fortran front-ends interface to a common back-end for code optimization and code generation components. Once your programs are compiled, linked and the executable file is produced, you can then execute or evaluate your application on the target system. The language your program is written in determines which compiler driver (or command-line) you should use to invoke the required compiler.

2.1 Using the x86 Open64 C/C++ Compiler

The x86 Open64 C compiler is invoked by the command:

```
$ opencc <input files>
```

The x86 Open64 C++ compiler is invoked by the command:

```
$ openCC <input files>
```

By default, input files to the C/C++ compiler will be preprocessed using the preprocessor. This default behavior can be overridden by specifying the switch '`-nocpp`'.

You can also specify other options or switches to the C/C++ compiler. See Section 3.1 [x86 Open64 Option Summary], page 27, for a list of these options and switches.

For example, to explicitly control the optimization level used by the compiler:

```
$ opencc -Ofast main.c foo.c
```

or

```
$ opencc -c -ipa main.c
$ opencc -c -ipa foo.c
$ opencc -ipa main.o foo.o
```

2.1.1 Pre-defined Macros

The C/C++ compiler predefines certain macros for the preprocessor. Follow these steps to find out what these macros are:

1. Compile your program with the switches -dD and -keep.
2. Look in the resulting .i file to see the list of pre-defined macros and their corresponding definitions.

For example:

```
$ opencc -dD -keep foo.c
$ cat foo.i
```

2.1.2 Pragmas

C/C++ pragmas enable the user to disable the effects of a command line option or apply new options to a selected part of a program without affecting the program as a whole. Pragmas are part of the C/C++ language, however, the meanings of the pragmas are implementation-specific.

2.1.2.1 Pragma `pack`

The syntax is:

```
#pragma pack (n)
```

This pragma specifies that the next structure should have each of their fields aligned to an alignment of n bytes, where n = 0, 1, 2, 4, 8, or 16. This only applies if its natural alignment is not smaller than n. When n = 0, the compiler returns to the default alignment for the subsequent struct definitions.

2.1.2.2 Unsupported GCC Extensions

The x86 Open64 C/C++ compiler supports majority of the C/C++ extensions supported by GCC version 4.2.0, except for the following extensions:

For C/C++:

- Complex integer data types (_Complex int)
- Structures that are generated/instantiated on the fly
- Indirect `goto` to an address of a label outside of the current block scope
- Nested functions
- Many of the `__builtin_` functions
- Attributes init_priority and java_interface
- Java-style exceptions

2.2 Using the x86 Open64 Fortran Compiler

The x86 Open64 Fortran compiler is invoked by the command:

```
$ openf90 <input files>
```

or

```
$ openf95 <input files>
```

By default, input files with suffixes of .F or .f are treated as fixed-form files, whereas input files with suffixes of .F90, .f90, .F95, or .f95 are treated as free-form files. This default behavior can be overridden by specifying the switches '`-fixedform`' or '`-freeform`'.

By default, input files with suffixes of .F, .F90, or .F95 will be preprocessed using the C preprocessor. This default behavior can be overridden by specifying the switches '`-cpp`' or '`-ftpp`'.

You can also specify other options or switches to the Fortran compiler. See Section 3.1 [x86 Open64 Option Summary], page 27, for a list of these options and switches.

For example, to explicitly control the optimization level used by the compiler:

```
$ openf90 -Ofast main.f sub.f
```

or

```
$ openf90 -c -ipa main.f
$ openf90 -c -ipa sub.f
$ openf90 -ipa main.o sub.o
```

2.2.1 Fixed-form and Free-form Fortran

Fixed form is the older form where character positions (columns) are reserved as follows:

Column	Description
1-5	Numerical labels or comments. – A 'C, !, * or /*' indicates that the rest of the line is a comment. – Comments are also indicated by a '!' in any column (except for the 6th column). Blank lines are treated as comments.
6	Line continuation. – Any character other than a blank means that the line is a continuation from the previous line. Open64 supports up to 499 continuation lines. The first 5 columns of a continuation line must be blank because these columns are for statement labels and these labels cannot appear on continuation lines.
7-72	Source code. – A ';' is used to separate multiple statements on a line, but cannot be the first non-blank character between positions 7-72.

Free form is much less restrictive and has the following features:

– No limitations on line length

– An '&' at the end of the line indicates that the next line is a continuation.

– A '!' anywhere on a line indicates a comment.

2.2.2 Pre-defined macros

The Fortran compiler predefines certain macros for the preprocessor. Depending on which preprocessor is used, different macros are pre-defined. Follow these steps to find out what these macros are:

If the C preprocessor is used (-cpp):

1. Compile your program with the switches -Wp,-dD -E.
2. The list of pre-defined macros and their corresponding definitions will be written to the stdout file.

For example:

```
$ openf90 -Wp,-dD -E foo.f -cpp > foo.i
$ cat foo.i
```

If the Fortran preprocessor is used ('`-ftpp`'):

Only the following macros are pre-defined:

```
LANGUAGE_FORTRAN 1
LANGUAGE_FORTRAN90 1
_LANGUAGE_FORTRAN90 1
unix 1
__unix 1
```

2.2.3 Fortran Modules

After a Fortran module is compiled, the compiler generates a file called '`foo.mod`' (where '`foo`' is the name of the module). By default this file is placed in the same directory where the compilation command is issued. This default behavior can be overridden by specifying the switch '`-module`'.

When compiling a Fortran file that uses the '`foo`' module, by default the Fortran compiler will look for the file '`foo.mod`' in the same directory where the compilation command is issued. This default behavior can be overridden by specifying the switch '`-I`*`directory`*'.

Files containing modules that are to be used by other files must be compiled before the other files. This can be accomplished by any or all of the following:

1. Modules must appear before their uses in the same source file.
2. Files containing modules that are to be used by other files must be compiled before the other files are compiled.
3. Files containing modules that are to be used by other files must appear before the other files in the same compilation command.

After the Fortran compiler compiles a file containing a module, the compiler generates an object file ('`.o`') as well as the module information file ('`.mod`'), even if the file contains nothing other than just the module. This object file must be linked with the rest of the program to generate an executable.

2.2.3.1 Linking a Main Program Contained in a Library

When you are working with many object files, it's convenient to place them all into a single library. So instead of specifying a long list of object files, you only have to specify the library when linking the program. The linker symbol for a main Fortran program is `MAIN__` rather than `main`. This results in the linker not automatically importing the program from a library when using openf90 to link to the program. The program will abort. To prevent your program from aborting, you need to tell the linker explicitly to import the symbol `MAIN__` (with two underscores):

```
$ openf90 -Wl, --undefined=MAIN__ objectlibrary.a
```

2.2.3.2 Module Error Messages

Error messages appear as the first line in the module, regardless of where in the module the error actually occurs. The real error is reported after this standard message, as shown in the following example.

Here is a program, '`hellow.f95`', which contains this module:

```
MODULE HELLO_WORLD
CONTAINS
SUBROUTINE HELLO_W( )
SPRINTZ *,"Hello, World!"
END SUBROUTINE HELLO_W
END MODULE HELLO_WORLD
```

Next compile the program containing the module:

```
$ openf95 hello_world.f95
     MODULE HELLO_WORLD
          ^
openf95-855 openf95: ERROR HELLO_WORLD, File = hello_world.f95, Line = 1,Column = 9
```

```
  The compiler has detected errors in module "HELLO_WORLD". No module information
  file will be created for this module.
     SPRINTZ *,"Hello, World!"
          ^
openf95-724 openf95: ERROR HELLO_W, File = hello_world.f95, Line = 4,Column = 10
  Unknown statement. Expected assignment statement but found "*" instead of "=" or "=>".

openf95: Open64 Fortran Version 4.2.2(f14) Mon May 11,2009 22:41:48
openf95: 6 source lines
openf95: 2 Error(s), 0 Warning(s), 0 Other message(s), 0 ANSI(s)
openf95: "explain openf95-message number" gives more information about each message
```

Note that the real error is pointed out after the first error on line 1 is reported.

2.2.4 Extensions

The x86 Open64 supports the following extensions to the Fortran standard:

- Promoting REAL and INTEGER Types
- Cray Pointers
- Directives (Prefetch and Change Optimization)

2.2.4.1 Promoting REAL and INTEGER Types

The following option is useful for porting from Cray code when integer and floating point data is 8-bytes long by default.

'`-r8`' Promotes the default representation for REAL type from 4 bytes to 8 bytes.

'`-i8`' Promotes the default representation for INTEGER type from 4 bytes to 8 bytes.

Consider the following when using this option:

- Always check for type mismatches with external libraries.
- The '`-r8`' and '`-i8`' flags do not affect variable declarations or constants that specify an explicit `KIND`. If a 4-byte default real or integer is passed into a subprogram that declares a `KIND=4` integer or real, the results will be incorrect.

  The following example shows the correct usage of `KIND`:

  ```
  VAR1 = KIND(1)
  ```

  or

  ```
  VAR2 = KIND(0.0d0)
  ```

  If you try to use `KIND = KIND(1)`, you will get the following error message:

  ```
  The left hand side of an assignment statement must be a variable or a
  function result
  ```

2.2.4.2 Cray Pointers

The Cray pointer provides a C-like pointer in Fortran for specifying dynamic objects. The Cray pointer differs from the Fortran pointer. Both the Cray and Fortran pointers use `POINTER`, but they are declared differently.

The Cray pointer is declared using:

```
POINTER ( <pointer>, <pointee> )
```

The "pointer" holds a memory address and the "pointee" is used to dereference the pointer.

The Fortran pointer is declared using:

```
POINTER :: [ <object)name> ]
```

The x86 Open64 uses the stricter Cray implementation of the Cray pointers. Specifically, x86 Open64 doesn't treat pointers exactly like integers. For example, if you use `p = ( (p+7) / 8) * 8` to align a pointer, the compiler flags this as an error.

2.2.4.3 Directives

Directives change the effects of certain command line options or default behavior of the compiler. While a command line option affects the entire source file being compiled, directives apply only to selected subroutines or loops. At the end of the selected portion of the program, the settings revert back to the command line options.

By default, directives within a file override the command line options. However, certain directives may have no effect unless additional options are present (for example, '`-mp`)'.

Following are options for changing this default:

Command line options to override directives:

```
-LNO:ignore_pragmas
```

Ignore directives contained within comments (such as `!$OMP` or `C*$* PREFETCH_REF`):

```
-no-directives
```

Scan the comments for directives:

```
-directives
```

Prefetch Directives

x86 Open64 supports the following prefetch directives:

`C*$* PREFETCH(`*N* `[,`*N*`])`

This directive specifies prefetching for each level of the cache.

N values:

0	Prefetching off (Default)
1	Prefetching on, but less aggressive than N=2.
2	Prefetching on, most aggressive. (Default when prefetch is on.)

Scope: Entire function containing the directive.

`C*$* PREFETCH_MANUAL(`*N*`)`

This directive specifies if manual prefetches (through directives) are respected or ignored.

N values:

0	Ignore manual prefetches
1	Respect manual prefetches

Scope: Entire function containing the directive.

`C*$* PREFETCH_REF_DISABLE=`*A* `[,` *size=num*`]`

This directive explicitly disables prefetching all references to array *A* in the current function. If enabled, the auto-prefetcher runs and ignores array *A*.

size=num This optional argument is the size of the array references in this loop in Kbyte. This value must be a constant. Use the size for volume analysis.

Scope: Entire function containing the directive.

`C*$* PREFETCH_REF=`*`array-ref`*`, [stride=[`*`str`*`] [,`*`str`*`]], [level=[lev] [,lev]], [kind=[rd/wr]], [size=[sz]]`

This directive generates a single prefetch instruction to the specified memory location. In the current loop-nest, this directive searches for array references that match the supplied reference.

'`Found`' Reference is connected to this prefetch node with the specified parameters.

'`Not Found`'
This prefetch node stays free-floating and is "loosely" scheduled.

The automatic prefetcher (if enabled) ignores all references to this array in this loop-nest.

If the size is supplied, the auto-prefetcher (if enabled) reduces the effective cache size by that amount.

The compiler attempts to issue one prefetch per stride iteration, but this cannot be guaranteed. Use redundant prefetches instead of transformations (such as inserting conditionals) that incur additional overhead.

This option uses the following arguments:

`array-ref`
Required. Array reference. For example: `A(i,j)`

`str` Optional. Prefetches every str iterations of this loop. Default = *1*.

`lev` Optional. The level in memory hierarchy to prefetch.

1 Prefetch from L2 to L1 cache.

2 Prefetch from memory to L1 cache.

The default is *lev=2*.

`rd/wr` Optional. Default is read/write.

`sz` Optional. The size in Kbytes of the array referenced in this loop. This value must be a constant.

Scope: No scope. Only generates a prefetch instruction.

Changing Optimization Using Directives

Optimization flags can be changed by using directives. The directive form is: `C*$* options <"list-of-options">`

You can specify any number of these directives inside the scope of the function. Each directive affects only the optimization of the entire function in which it is specified. You can add to the literal string an unlimited number of different options by separating the options

with a space and including the enclosing quotes. For the next function, all flags revert back to the settings specified in the compiler command line.

Following are some considerations for the options that are processed in this directive and their effects on the optimization:

- No warning or error is given for options that are not processed.
- Only options that affect optimizations are processed because these directive are processed only in the optimizing backend.
- The phase invocation of the backend components is not affected. For example, specifying `-O0` doesn't suppress the invocation of the global optimizer. However, the invoked backend phases will honor the specified optimization level.
- In addition to the optimization-level flags, only flags belonging to the following option groups are processed: `-LNO`, `-OPT` and `-WOPT`.

2.2.5 Varying Length Character Strings

x86 Open64 supports the ISO/IEC Standard 1539-2, which provides support for varying length character strings. You can download the module from the following location:

```
http://www.fortran.com/fortran/iso_varying_string.f95
```

2.2.6 Fortran 90 Dope Vector

Fortran provides constructs (for example, `ubound` and `size` for obtaining information about dynamically allocated objects, such as arrays and character strings. The compiler implements these constructs by maintaining information about the object in a data structure called a *dope vector*. Additional information can be found in the source distribution of the file: '`clibinc/cray/dopevec.h`'.

2.2.7 Bounds Checking

x86 Open64 can perform bounds checking on arrays. Use the `-C` option to generate an intermediate file:

```
$ openf95 -C foo.f90 -O foo
```

The generated code checks all array accesses to ensure they are within range of the array boundaries. Accesses that fall out of range, results in a warning at runtime:

```
$ ./foo
lib-4961 : WARNING
  Unable to find error message (check NLSPATH, file lib.cat)
 a[11]= 0
```

If you want the resulting program to abort on the first bounds check violation, set the environment variable `F90_BOUNDS_CHECK_ABORT` to `YES`. However, this will still result in a second bound warning as follows:

```
lib-4961 : WARNING
  Subscript 11 is out of range for dimension 1 for array
  'A' at line 5 in file 'simple_array.f' with bounds 1:10.
 a[11]= -1

lib-4961 : WARNING
  Subscript 12 is out of range for dimension 1 for array
  'A' at line 7 in file 'simple_array.f' with bounds 1:10.
 a[11]= -1
```

Enable array bounds checking only for debugging since it significantly slows code performance, i.e., disable in production code that is performance sensitive.

2.2.8 Pseudo-random Numbers

The pseudo-random number generator (PRNG) is a non-linear additive feedback PRNG with a 32-entry long seed table.

The period of the PRNG is approximately `16*((2**32) -1)`.

2.2.9 Fortran KINDs Compatibility

The x86 Open64 may have some compatibility issues with source code developed for other compilers because different compilers represent types in various ways.

The Fortran `KIND` attribute gives the user flexibility in specifying the precision or size of a type. Currently, Fortran uses `KINDS` to declare types. The following example shows the recommended and portable way to use `KIND` to inquire its value:

```
integer :: var3_kind = kind(0.0d0)
```

In practice, some users set the actual values in their programs:

```
integer :: var3_kind = 7
```

Using this way is unportable since some compilers use different values for the `KIND` of a double-precision floating-point value.

The x86 Open64 and most other compilers use the convention that the `KIND` value is the number of bytes in the type. For floating point numbers:

`KIND=4` 32-bit floating point

`KIND=8` 64-bit floating point

However, this convention is incompatible with unportable programs written using GNU Fortran, g77. For floating-point numbers, g77 uses:

`KIND=1` 32-bit single precision

`KIND=2` 64-bit double precision

For integer numbers, g77 uses:

`KIND=3` 1 btye

`KIND=4` 2 bytes

`KIND=1` 4 btyes

`KIND=2` 8 bytes

Currently, there is no compatibility flag for unportable g77 programs. It is best to following the recommended way for finding the actual KIND values.

If your programs uses -i8 or -r8, refer to the "Promoting REAL and INTEGER Types" section for more details.

2.2.10 Runtime I/O Compatibility

This section describes how the x86 Open64 compiler interacts with files generated by the Fortran I/O libraries on other systems. These files may contain data in different formats than that generated or expected by codes compiled by the Open64 compiler.

2.2.10.1 Using the I/O Complication Flags

To help with I/O, use the following two compilation flags:

`-byteswapio`
: This flag swaps bytes during I/O so that unformatted files on a little-endian processor are read and written in big-endian format (or vice versa).

`-convert conversion`
: This flag controls the swapping of bytes during I/O so that unformatted files on a little-endian processor are read and written in big-endian format (or vice versa). Use this option when compiling the Fortran main program.

  This flag takes one of the following arguments:

  . . . native - No conversion. Default

  . . . big_endian - Files are big endian.

  . . . little_endian - Files are little endian.

2.2.10.2 Reserved File Units

x86 Open64 reserves Fortran file units 5, 6, and 0.

2.3 Mixed Code

Generally when the argument data types and function return values agree, you can call a C/C++ function from Fortran and call a Fortran function from C/C++. Following are the calling considerations you should know about for mixed-code applications:

- C++ functions containing objects with constructors and destructors - It is not possible to call such functions from either C or Fortran unless you initialize the main program from a C++ program where the constructor and destructor are correctly initialized.
- Use extern "C" keyword to prevent "mangling" of function names - The C++ compiler mangles symbol names to implement overloading and adds to the data structures various information (such as virtual table pointer) that other languages cannot understand. When calling a C or Fortran function from C++, use the extern "C" keyword to declare the function in the C++ program. When calling a C++ function from C or Fortran, also use the extern "C" keyword to declare the C++ function.
- Use the _cplusplus macro to allow a program or header file to work for both C and C++.
- C++ member functions cannot be called from C or Fortran because C++ member functions cannot be declared `extern`.

2.3.1 Functions and Subroutines

Fortran, C, and C++ do not define functions and subroutines in the same way.

For a Fortran program calling a C or C++ function, the return value conventions are as follows:

- When a C/C++ function returns a value, call it from Fortran as a function.
- When a C/C++ function does not return a value, call it as a subroutine.

For a C/C++ program calling a Fortran function, the call should return a similar type. If the call is to a:

– Fortran subroutine
– Fortran `CHARACTER` function
– Fortran `COMPLEX` function

then call it from C/C++ as a function that returns void. The exception is when a Fortran subroutine has alternate returns. If this is the case, call this subroutine from a function returning `int` whose value is the value of the integer expression specified in the alternate `RETURN` statement.

2.3.2 Fortran Runtime Libraries

For applications with mixed Fortran, C, or C++ code, you have the option of invoking x86 Open64 with `opencc` or `openCC`, instead of `openf90` or `openf95`. If you do, you should always initialize the Fortran runtime libraries. Although standard Fortran I/O and most intrinsic functions will work without this initialization, the library is needed for runtime error messages, automatic stack sizing, and the intrinsics dealing with the command line arguments. The following example shows how to initialize the Fortran runtime libraries in mixed-code applications:

Example:

A large application that mixes Fortran code with code written in C or C++ and the main entry point to the application is from C or C++.

1. Optional - use `opencc` or `openCC` to link the application.
2. Manually add the Fortran runtime libraries to the link line as follows:
3. To link object files that were generated with `opencc` or `openCC` include the option '`-lstdc++`'.

```
$ opencc -o exe c_file.o fort_file.o -lfortran
```

2.3.3 Upper/Lower Case Conventions and Underscores

All Fortran symbol names are converted to lower case, whereas C and C++ are case sensitive. When you use mixed-code calling, you can either use all lower case for your C/C++ functions or use the Fortran compiler command with the option `-Mupcase`, so it will not convert symbol names to lower case.

`openf90` appends an underscore to Fortran global names (names of functions, subroutines and common blocks) when creating linker symbols as follows:

Procedure Name	**Linker Symbol**
myfunc	myfunc_
my_func	my_func__ (note: two trailing underscores)

However, `opencc` does not append any underscores to function names. You can match the Fortran convention by:

– Appending an underscore (`"x_"`) in C so that it matches the procedure name `"x"` in Fortran.

– Using the `-fdecorate` option to provide mapping from each Fortran name onto a linker symbol.

– Using the `-fno-underscoring` option. However, this option may create symbols that conflict with those in the Fortran and C runtime libraries.

– When Fortran calls a C/C++ function, use `C$PRAGMA C` in the Fortran program.

2.3.4 Data Types

You must be careful to match the data type of function/subroutine parameters and return values. Problematic data types to watch out for:

– Fortran `character` because it passes a pointer to the first character and appends an integer length-count argument to the end of the usual argument list.

– Fortran Cray pointers, declared with the pointer statement, correspond to C pointers. However, Fortran 90 pointers, declared with the `pointer` attribute, are unique to Fortran.

Table A shows how data types can be represented between Fortran and C/C++. Table B shows how Fortran `COMPLEX` type can be represented in C/C++.

Fortran Type	**C/C++ Type**	**Size (bytes)**
character x	char x	1
character*n x	char x[n]	n
real x	float x	4
real*4 x	float x	4
real*8 x	double x	8
double precision	double x	8
integer x	int x	4
integer*1 x	signed char x	1
integer*2 x	short x	2
integer*4 x	int x	4
integer*8 x	long long x	8
logical x	int x	4
logical*1 x	char x	1
logical*2 x	short x	2
logical*4	int x	4
logical*8	long long x	8

Table A: Fortran and C/C++ Data Type Compatibility

Fortran Type	C/C++ Type	Size (bytes)
complex x	typedef struct {float r, i;} complex;	8
	complex x;	8
complex*8 x	struct {float r,i;} x;	8
	float complex x;	8
double complex x	typedef struct {double dr,di;} complex;	16
	complex x;	16
complex*16 x	struct {double dr,di;} x;	16
	double complex x;	16

Table B: Fortran and C/C++ Representation of the `COMPLEX` Type

2.3.5 Passing Arguments and Returning Values

In Fortran, arguments are passed by reference (the address of the argument is passed, not the argument itself). In C/C++, arguments are passed by value, except for strings and arrays, which are passed by reference. The flexibility of the C/C++ language allows for ways around these differences, such as using the `&` and `*` operators in argument passing when C/C++ calls Fortran and in argument declarations when Fortran calls C/C++.

In Fortran, for strings declared as type `CHARACTER`, an argument for the string length is also passed to a calling function. The length argument is passed by value, not by reference. Open64 places the length argument(s) at the end of the parameter list, following the other formal arguments.

2.3.5.1 Passing by Value (`%VAL`)

When passing parameters from a Fortran subprogram to a C/C++ function, you can use the `%VAL()` intrinsic function to pass an argument by value. The following example shows a call passing the integer `i` and the logical `bvar` by value.

```
integer i
logical*1 bool
call call_c_by_val (%VAL(i), %VAL(bool))
```

2.3.5.2 Character Return Values

When a Fortran function returns a character, you need to add the following two arguments at the beginning of the C/C++ calling function's argument list:

- Address of the return character or characters
- Length of the return character

For example:

```
! Function returns a character
CHARACTER*(*) FUNCTION CHARFUNC(C1, I)
CHARACTER*(*) C
INTEGER I
END
```

The following C code sample shows where parameters tmp and 10 are supplied by the caller:

```
/* C declaration of Fortran function */
extern void charfunc_();
char return_array[5];
char ch[4];
int i;
charfunc_(return_array, 5, ch, &i, 4);
```

For a character value of constant length, for example:

```
CHARACTER*4 FUNCTION CHARFUNC()
```

you must still add the parameter representing the length, but it is not used. The value of the character function is not automatically NULL-terminated.

2.3.5.3 Complex Return Values

When a Fortran function returns a complex value, you need to add the following argument at the beginning of the C/C++ calling function's argument list:

– Address of the complex return value

The following example shows a Fortran function returning a complex value:

Fortran code sample:

```
COMPLEX FUNCTION FUNC(C, I)
 . . .
END
```

The following C code sample shows where parameters cplx is supplied by the caller:

```
extern void func_();
typedef struct {float r, i;} complex;
complex c;
int i;
func_(&c, &i);
```

2.3.5.4 Arrays and Structures

C/C++ arrays and Fortran arrays use different default initial array index values: C/C++ arrays start at 0 and Fortran arrays start at 1. You need to adjust your array comparisons accordingly.

C/C++ and Fortran arrays also use different storage methods. Fortran arrays are placed in memory in column-major order and C/C++ arrays use row-major order.

To make a Fortran 90 structure use the same layout as a C structure, try using the `sequence` keyword, although this may not work in every case. For arrays, limit the interface to the types of arrays provided in Fortran 77 because Fortran 90 introduced data structure information that C cannot understand. For example, an argument "a (5, 6)" or "a (n)" or "a (1:*)" (where n is a dummy argument) passes a simple pointer that corresponds to a C array. However, argument "a (:,:)" or an allocatable array or a Fortran 90 pointer array does not correspond to anything in C.

2.3.5.5 Fortran Named Common Blocks

Fortran named common blocks can be represented in C/C++ by a structure whose members correspond to the members of the common block. You must add an underscore to the name of the structure in C/C++. The following example shows the Fortran common block, C equivalent and C++ equivalent:

Fortran common block:

```
INTEGER I
COMPLEX C
DOUBLE PRECISION D
COMMON /COMMONBLOCK/ i, c, d
```

C equivalent:

```
extern struct {
  int i;
  struct {float r, i;} c;
  double d;
} commonblock_;
```

C++ equivalent:

```
/* extern "C" is not required for global or external data */

extern "C" struct {
  int i;
  struct {float r, i;} c;
  double d;
} commonblock_;
```

Accessing Common Blocks from C

Variables in Fortran 90 modules are grouped into common blocks. One block is for initialized data and the other block is for uninitialized data. You may use '`-fdecorate`' to access these common blocks from C, as shown in the following example:

Fortran Source Code (fprogram.f90):

```
module varmodule
  public
   integer :: var1
   double precision :: var2
   integer :: var3 = 11
   double precision :: var4 = 17.8
end module varmodule

program prog
  use varmodule
  var1 = 7
  var2 = 20.5
  call cfunction ()
end program prog
```

C Source Code (cprogram.c):

```
#include <stdio.h>
extern struct {
   int var1;
   double var2;
   } module_data;

extern struct {
   int var3;
   double var4;
   } module_data_initialized;

void cfunction ()
{
```

```
   printf ("%d %g\n", module_data.var1,
   module_data.var2);
   printf ("%d %g\n", module_data_initialized.var3,
   module_data_initialized.var4);
}

$ cat dfile
.data_init.in.varmodule module_data_initialized
.data.in.varmodule.in.varmodule module_data
cfunction cfunction
```

Use the following command to compile and execute:

```
$openf90 -fdecorate dfile fprogram.f90 cprogram.c
fprogram.f90:
cprogram.c:

$ ./a.out
11 17.8
7 20.5
```

2.3.6 Calls Between C and Fortran

The following example shows calls between C and Fortran.

C Source Code (c_code.c):

```
#include <stdio.h>
#include <alloca.h>
#include <string.h>

extern void function_(char *str, int *i, float *f, int str_len);

/* Calling Fortran from C */
void call_fortran()
{
   char *str = "Hello from call_fortran";
   int i = 221;
   float f = 8.1;
   function_(str, &i, &f, strlen(str));
}

/* Called from Fortran, passing arguments by reference */
void by_reference__(float *f, int *i,
        char *str1, int *bool, char *str2, int str1_len, int str2_len)
{

 /* A Fortran string has no null terminator, so make a local copy
  * and add a null character at the end. */

    printf("Arguments passed by reference\n");
    char *str1_copy = memcpy(alloca(str1_len + 1), str1, str1_len);
    char *str2_copy = memcpy(alloca(str2_len + 1), str2, str2_len);
    str1_copy[ str1_len] = str2_copy[ str2_len] = '\0';

    printf("float = %.1f, int = %d, bool = %d, "
            "str1_len = %d, str2_len = %d\n",
            *f, *i, *bool, str1_len, str2_len);
    printf ("str1 = '%s', str2 = '%s'\n", str1, str2);

fflush(stdout);          /* Flush output before switching languages */
```

```
call_fortran ();
}

/* Called from Fortran, passing arguments by value */

int by_value__(float f, int i)
{
    printf("Arguments passed by value\n");
    printf("float = %.1f, int = %d\n", f, i);
    fflush(stdout);
    return 4; /* true */
}
```

Fortran Source Code (f_code.f90):

```
program f_program
  implicit none

  interface
   subroutine by_reference(float, int, string1, bool, string2)
     real float
     integer int
     character* (*) string1, string2
     logical bool
    end subroutine by_reference

    logical function by_value(f, i)
          real f
          integer i
    end function by_value
  end interface

    logical l
    pointer (pusr, usr)
    character*32 usr
    ! Use decorate.txt to create a dummy mapping for C library calls
    ! else the compiler will complain about undefine reference to getlogin_
    integer*8 getlogin_dummy
    external getlogin_dummy
    intrinsic char

    ! Call a C function passing arguments by reference.
    call by_reference(6.2, 19, 'hello', .false., 'from f_program')

    ! Call a C function passing arguments by value.
    l = by_value( %val(12.9), %val(3) )
    write(6 , "(a,i)") "logical value returned = ", l

    ! "getlogin" is a libc function that returns "char*".
    ! When a C function returns a pointer, you must use a Cray pointer
    ! to receive the address and examine the data at that address,
    pusr = getlogin_dummy()
    write(6, "(3a)") "'", usr(1:index(usr, char(0)) - 1), "'"
end program f_program

! Called from C
subroutine function(str, i, f)
    implicit none
    character* (*) str
```

```
    integer i
    real f
    write(6, "(3a,i5,f5.1)") "'", str, "'", i, f
end subroutine function
```

This is the third file (decorate.txt):

```
getlogin_dummy getlogin
```

Compile and execute the three files (c_code.c, f_code.f90, and decorate.txt) with this command:

```
$ openf90 -Wall -fdecorate decorate.txt f_code.f90 c_code.c
$ ./a.out
Arguments passed by reference
float = 6.2, int = 19, bool = 0, str1_len = 5, str2_len = 14
str1 = 'hello', str2 = 'from f_program'
'Hello from call_fortran'  221  8.1
Arguments passed by value
float = 12.9, int = 3
logical value returned = 4
'john'
```

3 x86 Open64 Command Options

When you invoke x86 Open64, it normally does preprocessing, compilation, assembly and linking. The "overall options" allow you to stop this process at an intermediate stage. For example, the '`-c`' option says not to run the linker. Then the output consists of object files output by the assembler.

Other options are passed on to one stage of processing. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; most of these are not documented here, since you rarely need to use any of them.

Most of the command line options that you can use with Open64 are useful for C programs; when an option is only useful with another language (usually C++), the explanation says so explicitly. If the description for a particular option does not mention a source language, you can use that option with all supported languages.

See Section 3.4 [Invoking openCC], page 36, for a summary of special options for compiling C++ programs.

The `opencc` program accepts options and file names as operands. Many options have multi-letter names; therefore multiple single-letter options may *not* be grouped: '`-dr`' is very different from '`-d -r`'.

You can mix options and other arguments. For the most part, the order you use doesn't matter. Order does matter when you use several options of the same kind; for example, if you specify '`-L`' more than once, the directories are searched in the order specified.

Many options have long names which start with '`-f`' or with '`-W`'—for example, '`-funsafe-math-optimizations`', '`-Wformat`' and so on. Options of the form '`-fflag`' specify machine-independent flags. Most of these have both positive and negative forms; the negative form of '`-ffoo`' would be '`-fno-foo`'. This manual typically documents only one of these two forms, whichever one is not the default or the one you typically will use. You can figure out the other form by either removing '`no-`' or adding it.

See [Option Index], page 177, for an index to Open64's options.

3.1 Option Summary

Here is a summary of all the options, grouped by type. Explanations are in the following sections.

Overall Options
See Section 3.2 [Options Controlling the Kind of Output], page 32.

```
-c -S -E -o file -copyright -help -help:string
-keep -LIST: -LIST:all_options -LIST:notes -LIST:options -LIST:symbols
-r -show -show-defaults -show0 -showt -dumpversion -v -version -###
```

Directory Options
See Section 3.3 [Options for Directory Search], page 35.

```
-Idir -iquotedir -isystemdir -Ldir
-nostdinc -isysrootdir
```

C/C++ Language Options
See Section 3.5 [Options Controlling C/C++ Dialect], page 36.

```
-ansi -fgnu-keywords -fno-gnu-keywords -fms-extensions
-fno-builtin -fno-common -fprefix-function-name -fpack-struct
-fshort-double -fshort-enums -fshort-wchar -f[no-]signed-bitfields
-f[no-]signed-char -f[no-]strict-aliasing
-std=standard -traditional
```

C++ Language Options

See [Options Controlling C++ Dialect], page 40.

```
-fabi-version=N -f[no-]check-new -f[no-exceptions]
-fno-emit-exceptions -f[no-]gnu-exceptions -f[no-]rtti -fuse-cxa-atexit
```

Fortran Language Options

See Section 3.7 [Options to Control Language Features], page 43.

```
-ansi -auto-use module_name -byteswapio -colN
-convert conversion -d-lines -default64 -exten-source
-noextend-source -nog77mangle -pad-char-literals -rreal_spec -uname
```

Language Feature Options

See Section 3.7 [Options to Control Language Features], page 43.

```
-LANG:copyinout -LANG:formal_deref_unsafe -LANG:global_asm
-LANG:heap_allocation_threshold -LANG:IEEE_minus_zero -LANG:IEEE_save
-LANG:recursive -LANG:rw_const _LANG:short_circuit_conditionals
```

Language Independent Options

See Section 3.8 [Options which are Language Independent], page 45.

```
-alignN -backslash -f[no-]unwind-tables -finhibit-size-directive
-fpic fPIC -fno-ident -HP -HUGEPAGE -HP:bdt -HP:heap
-HUGEPAGE:bdt -HUGEPAGE:heap -ignore-suffix -opencc -no-opencc
-nobool -U name
```

Optimization Options

See Section 3.9 [Options that Control Optimization], page 48.

Options that Control Feedback Directed Optimizations

```
-fb-create -fb-opt -fb-phase -finstrument-functions
```

Options that Control Global Optimizations

```
-apo -mso -O0 -O1 -O2 -O3 -Os -Ofast
-WOPT:aggstr -WOPT:const_pre -WOPT:if_conv -WOPT:ivar_pre
-WOPT:mem_opnds -WOPT:retype_expr -WOPT:unroll -WOPT:val
```

Options that Control General Optimizations

```
-f[no-]fast-math -ffloat-store -fno-math-errno
-f[no-]unsafe-math-optimizations -m87-precision -noexpopt
-openmp -mp -chunk -OPT:alias -OPT:align_unsafe -OPT:asm_memory
-OPT:bb -OPT:cis -OPT:cyg_instr -OPT:div_split -OPT:early_mp
-OPT:early_intrinsics -OPT:fast_bit_intrinsics -OPT:fast_complex
-OPT:fast_exp -OPT:fast_io -OPT:fast_math -OPT:fast_nint
-OPT:fast_sqrt -OPT:fast_stdlib -OPT:fast_trunc -OPT:fold_reassociate
-OPT:fold_unsafe_relops -OPT:fold_unsigned_relops -OPT:goto
-OPT:IEEE_arithmetic -OPT:IEEE_NaN_inf -OPT:inline_intrinsics
-OPT:keep_ext -OPT:malloc_algorithm -OPT:malloc_alg
-OPT:Ofast -OPT:Olimit -OPT:pad_common -OPT:recip -OPT:reorg_common
-OPT:roundoff -OPT:ro -OPT:rsqrt -OPT:space -OPT:speculate
-OPT:transform_to_memlib -OPT:treeheight -OPT:unroll_analysis
-OPT:unroll_level -OPT:unroll_times_max -OPT:unroll_size
-OPT:wrap_around_unsafe_opt
```

Options that Control Interprocedural Optimizations

`-f[no-]implicit-inline-templates -f[no-]implicit-templates`
`-f[no-]inline -inline -INLINE -noinline -f[no-]inline-functions`
`-fkeep-inline-functions -INLINE:all -INLINE:aggressive`
`-INLINE:list -INLINE:must -INLINE:never -INLINE:none -INLINE:preempt`
`-ipa -IPA -IPA:addressing -IPA:aggr_cprop -IPA:alias -IPA:callee_limit`
`-IPA:cgi -IPA:clone_list -IPA:common_pad_size _IPA:cprop -IPA:ctype`
`-IPA:depth -IPA:dfe -IPA:dve -IPA:echo -IPA:field_reorder`
`-IPA:forcedepth -IPA:ignore_lang -IPA:inline -IPA:keeplight`
`-IPA:linear -IPA:map_limit -IPA:maxdepth -IPA:max_jobs -IPA:min_hotness`
`-IPA:multi_clone -IPA:node_bloat -IPA:plimit -IPA:pu_reorder`
`-IPA:relopt -IPA:small_pu -IPA:sp_partition -IPA:space -IPA:specfile`
`-IPA:use_intrinsic`

Options that Control Loop Nest Optimizations

`-LNO:apo_use_feedback -LNO:build_scalar_reductions -LNO:blocking`
`-LNO:blocking_size -LNO:fission -LNO:full_unroll -LNO:fu`
`-LNO:full_unroll_size -LNO:full_unroll_outer -LNO:fusion`
`-LNO:fusion_peeling_limit -LNO:gather_scatter -LNO:hoistif`
`-LNO:ignore_feedback -LNO:ignore_pragmas -LNO:local_pad_size`
`-LNO:minvariant -LNO:minvar -LNO:non_blocking_loads -LNO:oinvar`
`-LNO:opt -LNO:ou_prod_max -LNO:outer -LNO:outer_unroll_max`
`-LNO:ou_max -LNO:parallel_overhead -LNO:prefetch -LNO:prefetch_ahead`
`-LNO:prefetch_verbose -LNO:processors -LNO:sclrze -LNO:simd`
`-LNO:simd_reduction -LNO:simd_verbose -LNO:svr_phase1`
`-LNO:trip_count_assumed_when_unknown -LNO:trip_count -LNO:vintr`
`-LNO:vintr_verbose -LNO:interchange -LNO:unswitch`
`-LNO:unswitch_verbose -LNO:outer_unroll -LNO:ou -LNO:outer_unroll_deep`
`-LNO:ou_deep -LNO:outer_unroll_further -LNO:ou_further`
`-LNO:outer_unroll_max -LNO:ou_max -LNO:pwr2 -LNO:assoc`
`-LNO:cmp -LNO:cs -LNO:is_mem -LNO:ls -LNO:ps -LNO:tlb -LNO:tlbcmp`
`-LNO:tlbdmp -LNO:pf -LNO:prefetch`
`-LNO:prefetch_ahead -LNO:prefetch_manual`

Preprocessor Options

See Section 3.10 [Options Controlling the Preprocessor], page 78.

`-A predicate -A -predicate -C -cpp -dD -DI -dM -dN`
`-D`*`name`* `-D`*`var`* `-fe -f[no-]preprocessed -ftpp`
`-M -macro-expand -MD -MDtarget -MDupdate -MF -MG -MM -MMD -MP`
`-MQ -MT -nocpp -no-gcc -P -U`*`name`* `-Wp,`*`option`*

Assembler Option

See Section 3.11 [Passing Options to the Assembler], page 82.

`-fno-asm -Wa,`*`option`*

Linker and Library Options

See Section 3.12 [Options for Linking], page 82.

`-ar -c -S -E -f[no-]fast-stdlib -H -l`*`library`*
`-objectlist -nostartfiles -nodefaultlibs -nostdinc -nostdinc++`
`-nostdlib -shared -shared-libgcc -static-libgcc -static --static`
`-static-data -stdinc -symbolic -Xlinker -Wl,`*`option`*

Code Generation Options

See Section 3.13 [Options for Code Generation Conventions], page 86.

`-CG:cflow -CG:cmp_peep -CG:compute_to -CG:cse_regs -CG:gcm`
`-CG:inflate_reg_request -CG:load_exe -CG:local sched_alg -CG:locs_best`
`-CG:locs_reduce_prefetch -CG:locs_shallow_depth -CG:movnti -CG:p2align`
`-CG:p2align_freq -CG:post_local_sched -CG:pre_local_sched`
`-CG:prefer_legacy_regs -CG:prefetch -CG:ptr_load_use`
`-CG:push_pop_int_saved_regs -CG:sse_cse_regs -CG:unroll_fb_req`

```
-CG:use_prefetchnta -CG:use_test -GRA:home -GRA:optimize_boundary
-GRA:prioritize_by_density -GRA:unspill
```

Target Options

See Section 3.14 [Specifying Target Environment and Machine], page 90.

```
-TENV:frame_pointer -TENV:simd_dmask -TENV:simd_imask
-Tenv:simd_omask -TENV:simd_pmask -TENV:simd_umask
-TENV:simd_zmask -TENV:X
```

Machine Dependent Options

See Section 3.14.1 [Hardware Models and Configurations], page 91. *i386 and x86-64 Options*

```
-march -mtune -mcpu -m[no-]sse -m[no-]sse2
-m[no-]sse3 -m[no-]sse4a -m[no-]3dnow -m32 -m64 -mcmodel
```

Diagnostic Options

See Section 3.15 [Options to Control Diagnostic], page 94.

```
-C -clist -CLIST: -CLIST:dotc_file -CLIST:doth_file
-CLIST:emit_pfetch -CLIST:linelength -CLIST:show -flist
-FLIST: -FLIST:ansi_format -FLIST:emit_pfetch -FLIST:ftn_file
-FLIST:linelength -FLIST:show -f[no-]permissive -fullwarn
-pedantic-errors -trapuv -zerouv
```

Debugging Options

See Section 3.16 [Options for Debugging Your Program], page 97.

```
-dD -dI -dM -dN -fprofile-arcs -frandom-seed
-ftest-coverage -g -g0 -g1 -g2 -g3 -gdwarf-2 -gdwarf-20
-gdwarf-21 -gdwarf-22 -gdwarf-23 -p -pg -profile
```

Warning Options

See Section 3.17 [Options to Request or Suppress Warnings], page 99.

```
-w -Wall -Wbad-function-cast -W[no-]deprecated
-W[no-]disabled-optimization -W[no-]div-by-zero -W[no-]endif-labels
-W[no-]error -W[no-]float-equal -W[no-]import -W[no-]larger-than
-Wno-deprecated-declarations -woff -woffall -woffoptions -woffnum
-Wundef -Wno-undef -W[no-]uninitialized -W[no-]unknown-pragmas
-W[no-]unreachable-code -W[no-]unused -W[no-]unused_function
-W[no-]unused-label -W[no-]unused-parameter -W[no-]unused-value
-W[no-]unused-variable -W[no-]write-strings
```

Warning Options for C/C++ Only

```
-Waggregate-return -W[no-]cast-align -W[no-]char-subscripts
-W[no-]comment -W[no-]conversion -Wno-declaration-after-statement
-W[no-]format -W[no-]format-nonliteral -W[no-]format-security -w[no-]-id-
clash -W[no-]implicit -W[no-]implicit-function-declaration
-W[no-]implicit-int -W[no-]inline -W[no-]main -W[no-]missing-braces
-W[no-]missing-declarations -W[no-]missing-format-attribute
-W[no-]missing-noreturn -W[no-]missing-prototypes -W[no-]multichar
-W[no-]nested-externs -Wno-cast-qual -Wno-format-extra-args
-Wno-format-y2k -Wnonnull -Wno-non-template-friend
-W[no-]non-virtual-dtor
-Wno-pmf-conversions -W[no-]old-style-cast
-W[no]overloaded-virtual -W[no-]packed -W[no-]padded -W[no-]parentheses
-W[no-]pointer-arith -W[no-]redundant-decls -W[no-]redundant-decls
-W[no-]reorder -W[no-]return-type -W[no-]sequence-point -W[no-]shadow
-W[no-]sign-compare -W[no-]sign-promo -W[no-]strict-aliasing
-W[no-]strict-prototypes -W[no-]switch -Wswitch-default -Wswitch-enum
-W[no-]system-headers -W[no-]synth -W[no-]traditional -W[no-]trigraphs
```

Some command line options provide a group of suboptions. The x86 Open64 compiler supports a number of these group options, for example:

`-CG:` Code Generation

`-CLIST:` C Listing

`-FLIST:` Fortran Listing

`-GRA:` Global Register Allocator

`-HUGEPAGE:`
Huge Pages

`-INLINE:` Subprogram Inlining

`-IPA:` Interprocedural Analyzer

`-LANG:` Language

`-LIST:` Listing

`- LNO:` Loop Nest Optimizer

`-OPT:` General Optimizer

`-TENV:` Target Environment

`-WOPT:` Global Optimizer Modifier

Group options accept several suboptions allowing the user to specify a setting for each suboption. The general usage format is:

```
-GROUP_OPTION:question=answer
```

To specify multiple suboptions:

– either use colons to separate each suboption
– or specify multiple options on the command line.

The following command lines are equivalent:

```
% opencc -LIST:notes=ON -LIST:symbols=OFF foo.c
% opencc -LIST:notes=ON:symbols=OFF foo.c
```

Some *answer* to suboptions to group options are specified with a setting that either enables or disables the feature. To enable a feature, specify the suboption either alone or with *=1*, *=ON*, or *=TRUE*. To disable a feature, specify the suboption with either *=0*, *=OFF*, or *=FALSE*. The following command lines are equivalent:

```
% opencc -OPT:recip=ON:space=OFF:speculate=TRUE foo.c
% opencc -OPT:recip:space=0:speculate=ON foo.c
```

Note for brevity, this document uses only the *ON|OFF* settings to suboptions. The compiler also accepts *1|0* and *TRUE|FALSE* as settings.

Group options '`-INLINE:`' and '`-IPA:`' have noted differences for the '`GROUP_OPTION`' without any suboptions. Additionally specifying:

'`-INLINE`' is equivalent to '`-inline`'

'`-IPA`' is equivalent to '`-ipa`'

' `-clist`' is equivalent to '`-CLIST:=ON`'.

'`-flist`' is equivalent to enabling all the '`-FLIST`' options.

'`-HUGEPAGE`' is equivalent to '`-HUGEPAGE:heap=ON`'.

3.2 Options Controlling the Kind of Output

Compilation can involve up to four stages: preprocessing, compilation proper, assembly and linking, always in that order. x86 Open64 is capable of preprocessing and compiling several files either into several assembler input files, or into one assembler input file; then each assembler input file produces an object file, and linking combines all the object files (those newly compiled, and those specified as input) into an executable file.

For any given input file, the file name suffix determines what kind of compilation is done:

file`.c` C source code which must be preprocessed.

file`.cc`
file`.cxx`
file`.cpp`
file`.C` C++ source code which must be preprocessed. Note that in '`.cxx`', the last two letters must both be literally '`x`'. Likewise, '`.C`' refers to a literal capital C.

file`.i` C source code which should not be preprocessed. Note to preserve a *file*.i invoke the compiler with the '`-E`' command option.

file`.ii` C++ source code which should not be preprocessed.

file`.h` C/C++ header file to be turned into a precompiled header.

file`.hh`
file`.H` C++ header file to be turned into a precompiled header.

file`.f` Fixed format Fortran source code which should not be preprocessed.

file`.f90`
file`.f95` Freeform Fortran source code which should not be preprocessed.

file`.F` Fixed format Fortran source code which must be preprocessed (with the traditional preprocessor).

file`.F90`
file`.F95` Freeform Fortran source code which must be preprocessed (with the traditional preprocessor).

file`.s` Assembler code. Note to preserve a *file*.s invoke the compiler with the '`-S`' command option.

file`.S` Assembler code which must be preprocessed.

file`.o`
other An object file to be fed directly into linking. Any file name with no recognized suffix is treated this way.

file`.a` A static library of object files

file`.so` A library of shared object files

If you only want some of the stages of compilation, you can use filename suffixes to tell the compiler where to start, and one of the options '`-c`', '`-S`', or '`-E`' to say where the compiler is to stop.

`-c` Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file.

By default, the object file name for a source file is made by replacing the suffix '`.c`', '`.i`', '`.s`', etc., with '`.o`'.

Unrecognized input files, not requiring compilation or assembly, are ignored. Not to be used with '`-r`' since the '`-c`' option nullifies the '`-r`' option.

`-S` Stop after the stage of compilation proper; do not assemble. The output is in the form of an assembler code file for each non-assembler input file specified.

By default, the assembler file name for a source file is made by replacing the suffix '`.c`', '`.i`', etc., with '`.s`'.

Input files that don't require compilation are ignored.

`-E` Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code with line directives, which is sent to the standard output. Use '`-P`' to suppress line directives.

Input files which don't require preprocessing are ignored. The'`-E`' option nullifies the '`-nocpp`' option.

`-o` *file* Place output in file *file*. This applies regardless to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code.

If '`-o`' is not specified, the default is to put an executable file in '`a.out`', the object file for '*`source.suffix`*' in '*`source`*`.o`', its assembler file in '*`source`*`.s`', a precompiled header file in '*`source.suffix`*`.gch`', and all preprocessed C source on standard output.

`-copyright`
Display the copyright for the Open64 compiler.

`-help`
`-help:`*`string`*
Print (on the standard output) a description of the command line options understood by `opencc`, `openCC`, and `open95`.

If '`-help:`' is specified a description of the command line options that contain a given string is printed.

`-keep` Instructs the compiler to write all intermediate compilation files. Files written after final compilation are:

- the compiler generated preprocessed source code file, '`filename.i`'
- the compiler generated assembly language file, '`filename.s`'

Note if IPA is enabled and the assembly language file is required (i.e. '`filename.s`'), option '`-IPA:keeplight=OFF`' must be specified.

`-LIST:`*`question=answer`*
The '`-LIST:`' option group instructs the compiler to emit information which gets written to a listing file with the suffix '`.lst`'. The options in this group are:

`-LIST:=`*`ON`*`|`*`OFF`*
: Instructs the compiler to write the list file. The emitted data to the listing file includes a list of specified options. The default is '`-LIST:=ON`' if any '`-LIST:`*`question=answer`*' is specified, otherwise the default is '`-LIST:=OFF`'.

`-LIST:all_options=`*`ON`*`|`*`OFF`*
: Instructs the compiler to emit a list of supported options. The default is '`-LIST:all_options=OFF`'.

`-LIST:notes=`*`ON`*`|`*`OFF`*
: Specifying '`LIST:notes=OFF`' instructs the compiler **not** to insert a list of comments within the assembly listing that describes various actions taken by the compiler (e.g., software pipelining). Note enabling the generation of the assembly listing is a prerequisite to this option (e.g., by specifying '`-S`'). The default is '`-LIST:notes=ON`'.

`-LIST:options=`*`ON`*`|`*`OFF`*
: Instructs the compiler to list the command-line options directly or indirectly modified as a side effect of other options. The default is '`-LIST:options=OFF`'.

`-LIST:symbols=`*`ON`*`|`*`OFF`*
: Instructs the compiler to list information regarding the symbols (i.e. variables) managed by the compiler. The default is '`-LIST:symbols=OFF`'.

`-r` Instructs the compiler to generate a relocatable object file ('`.o`') and then stop.

`-show` Print the compilation phases as they execute with appropriate arguments and input/output files.

`-show-defaults`
: Print (on the standard output) a description of target specific command line options for each tool. Also prints the options in the `compiler.defaults` file.

  For C/C++ '`-show-defaults`' will also print the compatible `gcc` version.

`-show0` Print (on the standard output) the compilation phases without invoking the compiler.

`-showt` Print (on the standard output) the time required by each compilation phase.

`-dumpversion`
: Print the compiler version (for example, '`3.0`')—and don't do anything else.

`-v` Print (on the standard output) the compiler driver, preprocessor, compiler proper, and the commands specified to run on the compilation phases.

`-version` Print (on the standard output) the version number of the invoked compiler.

`-###` Like '`-v`' except the commands are not executed and all command arguments are quoted. This is useful for shell scripts to capture the driver-generated command lines.

3.3 Options for Directory Search

These options specify directories to search for header files, for libraries and for parts of the compiler. Note the space between the option and the directory name, *dir*, is not required (e.g., '`L` *dir*' is equivalent to '`L`*dir*').

`-I` *dir*
: Add the directory *dir* to the head of the list of directories to be searched for header files. This can be used to override a system header file, substituting your own version, since these directories are searched before the system header file directories. However, you should not use this option to add directories that contain vendor-supplied system header files (use '`-isystem`' for that). If you use more than one '`-I`' option, the directories are scanned in left-to-right order; the standard system directories come after.

  The '`-I`' is used for the following types of files:

  – File names that do **not** begin with a slash (/) character and are located on the `INCLUDE` statement of the Fortran source programs.
  – File names that do **not** begin with a slash (/) character and are located in the `#include` statement of a preprocessing directives.
  – Files specified by Fortran `USE` statements.

  If a standard system include directory, or a directory specified with '`-isystem`', is also specified with '`-I`', the '`-I`' option will be ignored. The directory will still be searched but as a system directory at its normal position in the system include chain. If you really need to change the search order for system directories, use the '`-nostdinc`' and/or '`-isystem`' options.

  The compiler searches for files in the following order:

  – first, in the directory that contains the input source file
  – second, in the directories specified by option '`I`*dir*'
  – third, in the standard '`/usr/include/`' directory.

`-iquote` *dir*
: Add the directory *dir* to the head of the list of directories to be searched for header files only for the case of '`#include "`*file*`"`'; they are not searched for '`#include <`*file*`>`', otherwise just like '`-I`'.

`-isystem` *dir*
: Search *dir* for header files, after all directories specified by -I but before the standard system directories. Mark it as a system directory, so that it gets the same special treatment as is applied to the standard system directories.

`-L` *dir*
: Add directory *dir* to the list of directories to be searched for '`-l`'. See ['`-l`*library*'], page 83. Note for XPG4 the order of searching for libraries named in '`-l`' is to look in the specified directory, *dir*, before looking in the default directory. Multiple instances of '`-L`' can be specified and are searched in the specified order left-to-right.

`-nostdinc`
: Do not search the standard system directories for header files. Only the directories you have specified with '`-I`' options (and the directory of the current file, if appropriate) are searched.

`-isysroot` *dir*

This option is like the '`--sysroot`' option, but applies only to header files. See the '`--sysroot`' option for more information.

3.4 Compiling C++ Programs

C++ source files conventionally use one of the suffixes '`.C`', '`.cc`', '`.cpp`', or '`.cxx`'; C++ header files often use '`.hh`' or '`.H`'; and preprocessed C++ files use the suffix '`.ii`'. x86 Open64 recognizes files with these names and compiles them as C++ programs even if you call the compiler the same way as for compiling C programs (usually with the name `opencc`).

However, the use of `opencc` does not add the C++ library. `openCC` is a program that calls x86 Open64 and treats '`.c`', '`.h`' and '`.i`' files as C++ source files instead of C source files unless '`-x`' is used, and automatically specifies linking against the C++ library. This program is also useful when precompiling a C header file with a '`.h`' extension for use in C++ compilations.

When you compile C++ programs, you may specify many of the same command-line options that you use for compiling programs in any language; or command-line options meaningful for C and related languages; or options that are meaningful only for C++ programs. See Section 3.5 [Options Controlling C Dialect], page 36, for explanations of options for languages related to C and C++.

3.5 Options Controlling C/C++ Dialect

The following options control the dialect of C (or languages derived from C, such as C++) that the compiler accepts:

`-ansi` In C mode, this is equivalent to '`-std=c89`'. In C++ mode, it is equivalent to '`-std=c++98`'.

This turns off certain features of x86 Open64 that are incompatible with ISO C90 (when compiling C code), or of standard C++ (when compiling C++ code), such as the `asm` and `typeof` keywords. It also enables the undesirable and rarely used ISO trigraph feature. For the C compiler, it disables recognition of C++ style '`//`' comments as well as the `inline` keyword.

The alternate keywords `__asm__`, `__extension__`, `__inline__` and `__typeof__` continue to work despite '`-ansi`'. You would not want to use them in an ISO C program, of course, but it is useful to put them in header files that might be included in compilations done with '`-ansi`'.

The '`-ansi`' option does not cause non-ISO programs to be rejected gratuitously. For that, '`-pedantic`' is required in addition to '`-ansi`'. See Section 3.17 [Warning Options], page 99.

The macro `__STRICT_ANSI__` is predefined when the '`-ansi`' option is used. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ISO standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things.

Functions that would normally be built in but do not have semantics defined by ISO C (such as `alloca` and `ffs`) are not built-in functions when '`-ansi`' is used.

`-fgnu-keywords`
`-fno-gnu-keywords`
'`-fgnu-keywords`' instructs the compiler to recognize `typeof` as a keyword. '`-fno-gnu-keywords`' specifies to **not** recognize `typeof` as a keyword, so that code can use this word as an identifier. You can use the keyword `__typeof__` instead. '`-ansi`' implies '`-fno-gnu-keywords`'. The default is '`-fno-gnu-keywords`'.

`-fms-extensions`
Accept some non-standard constructs used in Microsoft header files. This option disables pedantic warnings about constructs used in MFC, such as implicit int and getting a pointer to member function via non-standard syntax.

Some cases of unnamed fields in structures and unions are only accepted with this option.

`-fno-builtin`
Don't recognize built-in functions that do not begin with '`__builtin_`' as prefix. See Other built-in functions provided by x86 Open64 for details of the functions affected, including those which are not built-in functions when '`-ansi`' or '`-std`' options for strict ISO C conformance are used because they do not have an ISO standard meaning. Note the x86 Open64 C/C++ compiler does **not** support many of the `__builtin` functions.

Open64 normally generates special code to handle certain built-in functions more efficiently; for instance, calls to `alloca` may become single instructions that adjust the stack directly, and calls to `memcpy` may become inline copy loops. The resulting code is often both smaller and faster, but since the function calls no longer appear as such, you cannot set a breakpoint on those calls, nor can you change the behavior of the functions by linking with a different library. In addition, when a function is recognized as a built-in function, Open64 may use information about that function to warn about problems with calls to that function, or to generate more efficient code, even if the resulting code still contains calls to that function. For example, warnings are given with '`-Wformat`' for bad calls to `printf`, when `printf` is built in, and `strlen` is known not to modify global memory.

If you wish to enable built-in functions selectively when using '`-fno-builtin`', you may define macros such as:

```
#define abs(n)          __builtin_abs ((n))
#define strcpy(d, s)    __builtin_strcpy ((d), (s))
```

`-fno-common`
In C, allocate even uninitialized global variables in the data section of the object file, rather than generating them as common blocks. This has the effect that if the same variable is declared (without `extern`) in two different compilations, you will get an error when you link them. The only reason this might be useful

is if you wish to verify that the program will work on other systems which always work this way.

`-fprefix-function-name`
Instruct the compiler to attach a prefix to all function names.

`-fpack-struct[=n]`
Without a value specified, pack all structure members together without holes. When a value is specified (which must be a small power of two), pack structure members according to this value, representing the maximum alignment (i.e. objects with default alignment requirements larger than this will be potentially unaligned at the next fitting location).

Warning: the '`-fpack-struct`' switch causes the compiler to generate code that is not binary compatible with code generated without that switch. Additionally, it makes the code suboptimal. Use it to conform to a non-default application binary interface.

`-fshort-double`
Use the same size for `double` as for `float`.

Warning: the '`-fshort-double`' switch causes the compiler to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non-default application binary interface.

`-fshort-enums`
Allocate to an `enum` type only as many bytes as it needs for the declared range of possible values. Specifically, the `enum` type will be equivalent to the smallest integer type which has enough room.

Warning: the '`-fshort-enums`' switch causes the compiler to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non-default application binary interface.

`-fshort-wchar`
Override the underlying type for '`wchar_t`' to be '`short unsigned int`' instead of the default for the target.

Warning: the '`-fshort-wchar`' switch causes the compiler to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non-default application binary interface.

`-fsigned-bitfields`
`-fno-signed-bitfields`
These options control whether a bit-field is signed or unsigned, when the declaration does not use either `signed` or `unsigned`. By default, such a bit-field is signed, because this is consistent; the basic integer types such as `int` are signed types.

`-fsigned-char`
`-fno-signed-char`
Let the type `char` be signed, like `signed char`.

Each kind of machine has a default for what `char` should be. It is either like `unsigned char` by default or like `signed char` by default.

Ideally, a portable program should always use `signed char` or `unsigned char` when it depends on the signedness of an object. But many programs have been written to use plain `char` and expect it to be signed, or expect it to be unsigned, depending on the machines they were written for. This option, and its inverse, let you make such a program work with the opposite default.

The type `char` is always a distinct type from each of `signed char` or `unsigned char`, even though its behavior is always just like one of those two.

`-fstrict-aliasing`
`-fno-strict-aliasing`

Allows the compiler to assume the strictest aliasing rules applicable to the language being compiled. For C (and C++), this activates optimizations based on the type of expressions. In particular, an object of one type is assumed never to reside at the same address as an object of a different type, unless the types are almost the same. For example, an `unsigned int` can alias an `int`, but not a `void*` or a `double`. A character type may alias any other type.

Pay special attention to code like this:

```
union a_union {
  int i;
  double d;
};

int f() {
  a_union t;
  t.d = 3.0;
  return t.i;
}
```

The practice of reading from a different union member than the one most recently written to (called "type-punning") is common. Even with '`-fstrict-aliasing`', type-punning is allowed, provided the memory is accessed through the union type. So, the code above will work as expected. However, this code might not:

```
int f() {
  a_union t;
  int* ip;
  t.d = 3.0;
  ip = &t.i;
  return *ip;
}
```

Every language that wishes to perform language-specific alias analysis should define a function that computes, given a `tree` node, an alias set for the node. Nodes in different alias sets are not allowed to alias. For an example, see the C front-end function `c_get_alias_set`.

Enabled at levels '`-O2`', '`-O3`', '`-Os`'.

`-std=`*standard*

Determine the language standard. See Section 1.2 [Language Standards Supported by x86 Open64], page 3, for details of these standard versions. This option is currently only supported when compiling C or C++.

The compiler can accept several base standards, such as '`c89`' or '`c++98`', and GNU dialects of those standards, such as '`gnu89`' or '`gnu++98`'. By specifying a

base standard, the compiler will accept all programs following that standard and those using GNU extensions that do not contradict it. For example, '`-std=c89`' turns off certain features of GCC that are incompatible with ISO C90, such as the `asm` and `typeof` keywords, but not other GNU extensions that do not have a meaning in ISO C90, such as omitting the middle term of a `?:` expression. On the other hand, by specifying a GNU dialect of a standard, all features the compiler support are enabled, even when those features change the meaning of the base standard and some strict-conforming programs may be rejected. The particular standard is used by '`-pedantic`' to identify which features are GNU extensions given that version of the standard. For example '`-std=gnu89 -pedantic`' would warn about C++ style '`//`' comments, while '`-std=gnu99 -pedantic`' would not.

A value for this option must be provided; possible values are

'`c89`'
'`iso9899:1990`'
: Support all ISO C90 programs (certain GNU extensions that conflict with ISO C90 are disabled). Same as '`-ansi`' for C code.

'`iso9899:199409`'
: ISO C90 as modified in amendment 1.

'`c99`'
'`c9x`'
'`iso9899:1999`'
'`iso9899:199x`'
: Support all ISO C99 programs.

'`gnu89`' : GNU dialect of ISO C90 (including some C99 features). This is the default for C code.

'`gnu99`'
'`gnu9x`' : GNU dialect of ISO C99. The name '`gnu9x`' is deprecated.

'`c++98`' : The 1998 ISO C++ standard plus amendments. Same as '`-ansi`' for C++ code.

'`gnu++98`' : GNU dialect of '`-std=c++98`'.

The default is '`-std=gnu89`' for C code and '`-std=gnu++98`' for C++ code

`-traditional`
: Formerly, these options caused the compiler to attempt to emulate a pre-standard C compiler. They are now only supported with the '`-E`' switch. The preprocessor continues to support a pre-standard mode. See the GNU CPP manual for details.

This section describes the command-line options that are only meaningful for C++ programs; but you can also use most of the compiler options regardless of what language your program is in. For example, you might compile a file `firstClass.C` like this:

```
openCC -g -frepo -O -c firstClass.C
```

In this example, only '`-frepo`' is an option meant only for C++ programs; you can use the other options with any language supported by x86 Open64.

Here is a list of options that are *only* for compiling C++ programs:

`-fabi-version=N` (C++ Only)
: Use version *N* of the C++ ABI. Version 2 is the version of the C++ ABI that first appeared in g++ 3.4. Version 1 is the version of the C++ ABI that first appeared in g++ 3.2. Version 0 will always be the version that conforms most closely to the C++ ABI specification. Therefore, the ABI obtained using version 0 will change as ABI bugs are fixed.

  The default is '`-fabi-version=2`' or version 2.

`-fcheck-new` (C++ Only)
`-fno-check-new` (C++ Only)
: Check that the pointer returned by `operator new` is non-null before attempting to modify the storage allocated. This check is normally unnecessary because the C++ standard specifies that `operator new` will only return `0` if it is declared '`throw()`', in which case the compiler will always check the return value even without this option. In all other cases, when `operator new` has a non-empty exception specification, memory exhaustion is signalled by throwing `std::bad_alloc`. See also '`new (nothrow)`'. '`-fno-check-new`' instructs the compiler to **not** check the result of `operator new` for null.

`-fno-emit-exceptions` (C++ Only)
: Enables exception handling, but does not generate code needed to raise/catch exceptions. This option allows the compiler to accept exceptions as part of the C++ dialect but in effect asserts that these exceptions will not actually be raised at runtime.

`-fexceptions` (C++ Only)
`-fno-exceptions` (C++ Only)
`-fgnu-exceptions` (C++ Only)
`-fno-gnu-exceptions` (C++ Only)
: Enable exception handling. Generates extra code needed to propagate exceptions. For some targets, this implies the compiler will generate frame unwind information for all functions, which can produce significant data size overhead, although it does not affect execution. If you do not specify this option, Open64 will enable it by default for languages like C++ which normally require exception handling, and disable it for languages like C that do not normally require it. However, you may need to enable this option when compiling C code that needs to interoperate properly with exception handlers written in C++. You may also wish to disable this option if you are compiling older C++ programs that don't use exception handling.

`-frtti` (C++ Only)
`-fno-rtti` (C++ Only)
: When specifying '`-frtti`' the compiler will emit runtime type information. '`-fno-rtti`' disables generating information about every class with virtual functions for use by the C++ runtime type identification features ('`dynamic_cast`' and '`typeid`'). If you don't use those parts of the language, you can save some space by using this flag. Note that exception handling uses the same information, but it will generate it as needed. The '`dynamic_cast`' operator can still

be used for casts that do not require runtime type information, i.e. casts to `void *` or to unambiguous base classes.

`-fuse-cxa-atexit` (C++ Only)
Register destructors for objects with static storage duration with the `__cxa_atexit` function rather than the `atexit` function. This option is required for fully standards-compliant handling of static destructors, but will only work if your C library supports `__cxa_atexit`.

3.6 Options Controlling Fortran Dialect

The following options control the dialect of Fortran that the compiler accepts:

`-ansi` Instructs the compiler to emit messages regarding constructs that violate Fortran syntax rules and constraints. Including messages about obsolescent and deleted features. This option disables all nonstandard intrinsic functions and subroutines. Note specifying '`-ansi`' implies option '`-ffortran2003`'and when used concurrently with '`-fullwarn`' causes all messages to be generated (i.e. regardless of level).

`-auto-use` *`module_name`*`[,`*`module_name`*`]...`
Instruct the compiler to act as if a `USE` *module_name* statement was entered in the Fortran source code for each *module_name*. The compiler inserts `USE` statements in every program unit and interface body in the compiled source code. For example,

```
openf95 -auto-use mpi_interface
```

or

```
openf95 -auto-use shmem_interface
```

Note in some situations using '`-auto-use`' can increase compiler time.

`-byteswapio`
'`-byteswapio`' swaps bytes during I/O so that unformatted files on a little-endian processor are read and written in big-endian format (or vice versa.) Use the option when compiling the Fortran main program. Note the '`-byteswapio`' option affects record headers as well as data in sequential unformatted files. Setting the environment variable `FILENV` during program execution will supersede the compiled-in code generated option in favor of the choice established by the `assign` command.

`-col`*`N`* Specifies the line width for fixed-format source lines. Set *N* to 72, 80, or 120.

`-convert` *`conversion`*
'`-convert` *`conversion`*' controls the swapping of bytes during I/O so that unformatted files on a little-endian processor are read and written in big-endian format (or vice versa). Use the option when compiling the Fortran main program. Note the '`-byteswapio`' option affects record headers as well as data in sequential unformatted files. Setting the environment variable `FILENV` during program execution will supersede the compiled-in code generated option in favor of the choice established by the command assign. The option supports three arguments

native No conversion

little_endian
Files are little-endian

big_endian
Files are big-endian

The default is '`-convert native`'.

`-d-lines` Instruct the compiler to insert a `D` in column 1 of compiled lines.

`-default64`
The compiler sets the sizes of default integer, real, logical, and double precision objects. '`-default64`' causes the following options to go into effect: '`-r8`' and '`-i8`'. Note calling a function in a specialized library requires that its 32-bit (or 64-bit) entry point be specified when 32-bit (or 64-bit) data is being used.

`-extend-source`
`-noextend-source`
The line length for fixed-format source files are set to 132 character-per-line. The default for fixed-format lines are 72 characters-per-line. See ['`-colN`'], page 42, for more information on controlling line length.

`-nog77mangle`
Fortran symbol names are modified by the compiler by appending an underscore to the symbol name, e.g., symbol name `foo` in the source file becomes `foo_` in the object file.

If the symbol name includes an underscore then the compiler will append a second underscore to the symbol name in the object file, e.g., `foo_` and `foo_bar` in the source file becomes `foo__` and `foo_bar__` in the object file, respectively. '`nog77mangle`' suppresses the appending of the second underscore.

`-pad-char-literals`
Extend the length (i.e., by padding with blanks) of all character literal constants to the size of the default integer type and that are passed as actual arguments.

`-rreal_spec`
Specifies the default `KIND` specification for real values.

`-r4` Use `REAL(KIND=4)` for real variables and `COMPLEX(KIND=4)` for complex variables.

`-r8` Use `REAL(KIND=8)` for real variables and `COMPLEX(KIND=8)` for complex variables.

The default is '`-r4`'.

`-uname` Instructs the compiler to assign the default type of the variable, *name*, to be undefined rather then using default Fortran 90 rules.

3.7 Options to Control Language Features

The options described below can be used to control the features of the C/C++ or Fortran language.

`-LANG:`*question=answer*

Options in the '`-LANG:`' group can be used to control the set of features that are supported. For example, to compile code that does not conform with the Standard in one way or another. Note it may not always be possible, however, to link together object files, some of which have been compiled with a feature enabled and others with it disabled.

`-LANG:copyinout=`*ON*`|`*OFF*

If an array section is passed as an argument in a call, the compiler is instructed to copy the array section into a temporary array that will be passed as the argument in the call. '`-LANG:copyinout`' optimizes the accessing of array arguments by improving argument locality. Note the flag helps regulate the aggressiveness of this optimization and is mainly suited to Fortran code. When specifying global optimization '`-O2`' or higher '`-LANG:copyinout=ON`'. The default is '`-LANG:copyinout=OFF`'.

`-LANG:formal_deref_unsafe=`*ON*`|`*OFF* (Fortran Only)

The compiler is instructed that it is unsafe to attempt any optimizations regarding the dereference of a formal parameter. The default is '`-LANG:formal_deref_unsafe=OFF`'.

`-LANG:global_asm=`*ON*`|`*OFF*

The compiler's assembler is instructed to allocate objects to sections if the program includes a file-scope assembly statement. '`-LANG:global_asm=ON`' causes some alignment optimizations to be suppressed allowing the allocations performed by the compiler to be compatible with the allocations in the assembly statement. The default is '`-LANG:global_asm=OFF`'.

`-LANG:heap_allocation_threshold=`*size*

Specifies the threshold when determining if to allocate an automatic array or compiler temporary on the heap rather than on the stack. Parameter *size* is in bytes and sets the threshold. Setting *size* to -1 implies all objects are placed on the stack. Setting *size* to 0 implies all objects are placed on the heap. The default is '`-LANG:heap_allocation_threshold=-1`' for maximum performance.

`-LANG:IEEE_minus_zero=`*ON*`|`*OFF* (Fortran only)

Instructs the compiler to acknowledge negative floating-point zeroes (i.e. -0.0). '`-LANG:IEEE_minus_zero=OFF`' disables the intrinsic function `SIGN(3I)`, suppressing the minus sign on zero. This deters problems created by optimizations and hardware instructions that return a negative floating-point zero result from a positive floating-point zero value. Note use the '`-z`' option with the `assign` command to print the minus sign of a negative floating-point zero. The default is '`-LANG:IEEE_minus_zero=OFF`'.

`-LANG:IEEE_save=`*ON*`|`*OFF* (Fortran only)
When a procedure accesses a standard IEEE intrinsic module with a `USE` statement, then upon entry the floating-point flags, halt mode, and rounding mode must be saved. When exiting the halt and rounding mode must be restored and the saved floating-point flags must be logically `OR` with the current floating-point flags. To improve runtime set the option to *OFF*. The default is '`-LANG:IEEE_save=OFF`'.

`-LANG:recursive=`*ON*`|`*OFF* (Fortran only)
When set to *ON* a statically allocated local variable can be referenced or modified by a recursive procedure call. The statically allocated local variable must be stored in memory before making a call and reloaded afterward.

When set to *OFF* the compiler can safely assume a statically allocated local variable will not be referenced or modified by a procedure call and can optimize more aggressively.

In either mode, the compiler supports a recursive stack-based calling sequence. The difference is in the optimization of statically allocated local variables. The default is '`-LANG:recursive=OFF`'.

`-LANG:rw_const=`*ON*`|`*OFF* (Fortran Only)
Instructs the compiler to handle constant parameter as either read-only or read-write. If the compiler is instructed to handle a constant parameter as read-write, then extra code must be generated to accommodate the possibility of the constant parameter being changed in the called function. Note when turned *OFF* the compiler generates more efficient code but segmentation faults will occur when writing to the constant parameter. The default is '`-LANG:rw_const=OFF`'.

`-LANG:short_circuit_conditionals=`*ON*`|`*OFF* (Fortran Only)
The compiler is instructed to manage the logical `.AND.` and `.OR.` by applying a short-circuit methodology to the second operand. Note if determined unnecessary, the compiler will **not** evaluate the second operand even if problems occur in addition to the desired compilation effect. The default is '`-LANG:short_circuit_conditionals=ON`'.

3.8 Options which are Language Independent

`-align`*N* Arranges for the data to be aligned on common blocks to a specified boundary. Selections for '`-align`*N*' are as follows:

`-align32` Align common blocks of data on 32-bit boundaries.

`-align64` Align common blocks of data on 64-bit boundaries.

Objects smaller than the specified alignment (i.e. 32-bit or 64-bit) are aligned on boundaries according to the object size. For example if '`-align64`' is selected:

- object sizes less than 64-bits but at least 32-bits are aligned on 32-bit boundaries.
- object sizes less than 32-bits but at least 16-bits are aligned on 16-bit boundaries.
- object sizes less than 16-bits are aligned on 8-bit boundaries.

`-backslash`
: Instructs the compiler to consider a backslash as a normal character instead of an escape character. Note specifying '`-backslash`' instructs the compiler not to pass the code through the preprocessor.

`-funwind-tables`
`-fno-unwind-tables`
: Similar to '`-fexceptions`', except that it will just generate any needed static data, but will not affect the generated code in any other way. You will normally not enable this option; instead, a language processor that needs this handling would enable it on your behalf. The default is '`-fno-unwind-tables`'.

`-finhibit-size-directive`
: Don't output a `.size` assembler directive, or anything else that would cause trouble if the function is split in the middle, and the two halves are placed at locations far apart in memory. This option is used when compiling '`crtstuff.c`'; you should not need to use it for anything else.

`-fpic`
: Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of the compiler; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-specific maximum size, you get an error message from the linker indicating that '`-fpic`' does not work; in that case, recompile with '`-fPIC`' instead.

  When this flag is set, the macros `__pic__` and `__PIC__` are defined to 1.

`-fPIC`
: If supported for the target machine, emit position-independent code, suitable for dynamic linking and avoiding any limit on the size of the global offset table.

  Position-independent code requires special support, and therefore works only on certain machines.

  When this flag is set, the macros `__pic__` and `__PIC__` are defined to 2.

`-fno-ident`
: Instruct the compiler to disregard the `#ident` directive.

`-HP:`*question*`=`*answer*`[,argument=`*N*`,...]`
`-HUGEPAGE:`*question*`=`*answer*`[,argument=`*N*`,...]`
`-HP`
`-HUGEPAGE`
: The compiler supports command line options to use 2 MByte huge pages for heap and "bdt", where "bdt" stands for bss, data and text segments. This

option group is operating system (OS) dependent, refer to the release notes for more information on which OS versions are supported.

A mixed usage of huge pages and small pages is supported for heap allocation when huge page is not available. A mixed usage of huge pages and small pages for bss, data and text segments is not supported.

Currently, the compiler requires a modified version of the `libhugetlbfs` library functions, see the release notes or supported versions. Both a static link and dynamic link to the modified library functions are supported for huge page heap allocation, but only dynamic link is supported for huge page "bdt" mapping.

`-HP:bdt=`*`size`*
`-HUGEPAGE:bdt=`*`size`*

The compiler is instructed to use huge pages for bss, data, and text segments (a.k.a, bdt). A mixed usage of huge pages and small pages is currently **not** supported for option '`-HUGEPAGE:bdt=`*`size`*' (i.e. bss, data, and text). The sub-option can be set to:

2m Instructs the compiler to use 2 MByte huge pages for bss, data, and text segments.

`-HP:heap=`*`size`*`[,argument=`*`N`*`,...]`
`-HUGEPAGE:heap=`*`size`*`[,argument=`*`N`*`,...]`

The compiler is instructed to use huge pages for the heap segment. A mixed usage of huge pages and small pages is currently supported for '`-HUGEPAGE:heap=`*`size`*'. The sub-option can be set to:

2m Instructs the compiler to use 2 MByte huge pages for bss, data, and text segments.

Currently the huge page sub-option for the heap supports one argument, '`-HUGEPAGE:heap=2m,limit=`*`N`*', where the value *N* is used to set the upper bound and represents a 32-bit integral number. If the *limit* argument is **not** specify or *N* is set to a negative number then **no** user-imposed limit is set and huge page usage is only limited by the resources on the target machine. For example, sub-options '`-HP:heap=2m`' and '`-HP:heap=2m,limit=-1`' are equivalent and specify that the upper bound is limited by the target resources. If the value of *N* is set to zero the huge page usages for heap is suppressed.

`-HP`
`-HUGEPAGE`

'`-HUGEPAGE`' and '`-HP`' are abbreviated forms for sub-options '`-HUGEPAGE:heap=2m`' and '`-HP:heap=2m`', respectively. For example:

```
opencc -HP foo.c
```

is equivalent to

```
opencc -HP:heap=2m foo.c
```

Examples for using huge pages are:

```
opencc -HUGEPAGE:bdt=2m -HUGEPAGE:heap=2m,limit=850 -o foo foo.c
```

or in a condensed form

```
opencc -HP:bdt=2m:heap=2m,limit=850 -o foo foo.c
```

Note the huge page library, `libhugetlbfs`, is not NUMA sensitive. It assumes that huge pages on all memory nodes in the system are available to all processes. It is recommended to use the *limit* argument to impose an upper bound on the huge page usage per process to avoid the situation that one process consumes all huge page resources and starves the rest.

If "numctrl -m" is used to bind a process to a specific memory node, then the memory node must have enough huge pages to meet the demand. Therefore it is required to configure sufficient huge pages for all threads and all processes. Note the huge page library, `libhugetlbfs`, is not multithread safe.

`-ignore-suffix`

The '`-ignore-suffix`' flag instructs the compiler to ignore the suffix of the input source files. The language of the source file is designated by the compiler driver. Specifying '`-ignore-suffix`' forces the compiler driver `opencc` to invoke the C compiler, `openCC` to invoke the C++ compiler, and `openf95` to invoke the Fortran compiler. By default the language of the source file is determined by the suffixes of the file (i.e. '`.c`', '`.cpp`', '`.C`', '`.cxx`', '`.f`', '`.f90`', and '`.s`').

`-opencc`

`-no-opencc`

'`-opencc`' instructs the compiler to define the __OPEN64__ macro and other predefined preprocessor macros. '`-no-opencc`' instructs the compiler to disable these macros.

`-nobool` Instruct the compiler to **not** allow boolean keywords.

`-U` *name* Instructs the compiler to remove any initial definition of *name*.

3.9 Options That Control Optimization

These options control various sorts of optimizations.

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make any debugging session produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

The compiler performs optimization based on the knowledge it has of the program. Optimization levels '`-O`' and above, in particular, enable *unit-at-a-time* mode, which allows the compiler to consider information gained from later functions in the file when compiling a function. Compiling multiple files at once to a single output file in *unit-at-a-time* mode allows the compiler to use information gained from all of the files when compiling each of them.

Not all optimizations are controlled directly by a flag. Only optimizations that have a flag are listed.

3.9.1 Options that Control Feedback Directed Optimizations

Feedback directed optimizations (FDO) is used by the x86 Open64 compiler to improve performance. The program under development must be compiled at least twice. The first compilation generates an executable which contains extra instrumentation library calls required to gather feedback information. At runtime, this specified instrumented executable is used to gather the required profile information about the program. The profile data is then used in subsequent compilations to perform the necessary transformations to produce optimum code.

`-fb-create` *filename*

Instructs the compiler to generate an instrumented executable program from the source code under development. The instrumented executable produces feedback data files at runtime using an example dataset. *filename* specifies the name of the feedback data file generated by the instrumented executable.

```
opencc -O2 -ipa -fb-create fbdata -o foo foo.c
```

'`fbdata`' will contain the instrumented feedback data from the instrumented executable '`foo`'. The default is '`-fb-create`' is disabled.

`-fb-opt` *filename*

Instructs the compiler to perform a feedback directed compilation using the instrumented feedback data produced by the '`-fb-create`' option.

```
opencc -O2 -ipa -fb-opt fbdata -o foo foo.c
```

The new executable, '`foo`', will be optimized to execute faster, and will **not** include any instrumentation library calls. Note the same optimization flags specified when creating the instrumented data file with the '`-fb-create`' must be specified when invoking the compiler with the '`-fb-opt`' option. Otherwise, the compiler will emit checksum errors. The default is '`-fb-opt`' disabled.

`-fb-phase=`*0*|*1*|*2*|*3*|*4*

Specifies the compilation phase when the collection of instrumentation data is to be performed. The values for option '`-fb-phase`' must be in the range of *0* to *4* and is used in conjunction with '`-fb-create`'. Note the value *0* indicates the initial phase, which is at the output of the preprocessor (i.e. after the front-end processing). The default is '`-fb-phase=0`'.

`-finstrument-functions`

Generates instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions will be called with the address of the current function and its call site. (On some platforms, `__builtin_return_address` does not work beyond the current function, so the call site information may not be available to the profiling functions otherwise.)

```
void __cyg_profile_func_enter (void *this_fn,
                               void *call_site);
void __cyg_profile_func_exit  (void *this_fn,
                               void *call_site);
```

The first argument is the address of the start of the current function, which may be looked up exactly in the symbol table.

This instrumentation is also done for functions expanded inline in other functions. The profiling calls will indicate where, conceptually, the inline function is entered and exited. This means that addressable versions of such functions must be available. If all your uses of a function are expanded inline, this may mean an additional expansion of code size. If you use '`extern inline`' in your C code, an addressable version of such functions must be provided. (This is normally the case anyway, but if you get lucky and the optimizer always expands the functions inline, you might have gotten away without providing static copies.)

A function may be given the attribute `no_instrument_function`, in which case this instrumentation will not be done. This can be used, for example, for the profiling functions listed above, high-priority interrupt routines, and any functions from which the profiling functions cannot safely be called (perhaps signal handlers, if the profiling routines generate output or allocate memory).

Note specifying '`-finstrument-functions`' implies '`-OPT:cyg_instr=3`', for more details See ['-OPT:cyg_intsr=3'], page 55.

3.9.2 Options that Control Global Optimizations

`-apo` Instructs the compiler to automatically transform sequential code into parallel code. The compiler will only transform blocks of code that will demonstrate a speed-up to the runtime (i.e. execute faster) on a multiprocessor target.

`-mso` Instructs the compiler to perform aggressive optimizations that are likely to improve the scalability of an application running on a system with multi-core processors. In particular, these optimizations may target machine resources that are shared among the multiple cores of a processor, e.g. memory bandwidth, shared L3 cache, etc.

`-O0|1|2|3|s`
Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

- `-O0` No optimizations are performed.
- `-O1` Perform minimal local optimizations on sections of straight-line code (basic blocks) only. Examples of such optimizations are instruction scheduling and some peephole optimizations. These optimizations do not usually have any noticeable impact on compilation time.
- `-O2` Perform extensive global optimizations. Examples of such optimizations are control flow optimizations, partial redundancy elimination and strength reduction. These optimizations can very often reduce the execution time of the compiled program significantly, but they may do so at the expense of increased compilation time. This is the default level of optimization.

`-O3` Perform all the optimizations at the '`-O2`' level as well as many more aggressive optimizations. Examples of such aggressive optimizations are loop nest optimizations and generation of prefetch instructions. Although these more aggressive optimizations can significantly sped up the run time execution of the compiled program, in rare cases they may not be profitable and may instead lead to a slow down. Also, some of these more aggressive optimizations may affect the accuracy of some floating point computations.

`-Os` Optimize for size. '`-Os`' enables all '`-O2`' optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size.

If you use multiple '`-O`' options, with or without level numbers, the last such option is the one that is effective. Level *2* is assumed if no value is specified (i.e. '`-O`'). The default is '`-O2`'.

`-Ofast` Uses a selection of optimizations in order to maximize performance. Specifying '`-Ofast`' is equivalent to

```
-O3
-ipa
-OPT:Ofast
-fno-math-errno
-ffast-math
```

These optimization options are generally safe. Floating-point accuracy may be affected due to the transformation of the computational code. Note that the interprocedural analysis option, '`-ipa`', specifies limitations on how libraries and object files ('`.o`' files) are built.

`-WOPT:`*question=answer*

This group of options controls the effect the global optimizer has on the program. '`-WOPT:`' only influences global optimizations specified by '`-O2`' or above.

`-WOPT:aggstr=`*N*

Option '`-WOPT:aggstr`' regulates the aggressiveness of the compiler's scalar optimizer when performing strength reduction optimizations. Strength reduction is the substitution of induction expressions within a loop with temporaries that are incremented together with the loop variable. The value *N* specifies the maximum number of induction expressions being replaced. Select positive integers only for variable *N*. Setting $N=0$ tells the scalar optimizer to use strength reduction for non-trivial induction expressions. Note specifying very aggressive strength reductions may prompt additional temporaries increasing register pressure and resulting in excessive register spills that decrease performance. The default is '`-WOPT:aggstr=11`'.

`-WOPT:const_pre=`*ON*`|`*OFF*

'`-WOPT:const_pre=ON`' instructs the compiler to perform the loading of registers using placement optimizations. The default is '`-WOPT:const_pre=ON`'.

`-WOPT:if_conv=`*0*|*1*|*2*

'`-WOPT:if_conv`' instructs the compiler to transform simple `IF` statements to conditional move instructions. '`-WOPT:if_conv`' has three settings:

0 Disables this optimization

1 Specifies conservative `IF` statement transformations. The context surrounding the `IF` statement is used in the transformation decision.

2 Use aggressive `IF` statement transformations. Perform the `IF` statement transformation regardless of the surrounding context.

The default is '`-WOPT:if_conv=1`'.

`-WOPT:ivar_pre=`*ON*|*OFF*

'`-WOPT:ivar_pre=ON`' instructs the compiler to use partial redundancy elimination of indirect loads in the program. The default is '`-WOPT:ivar_pre=ON`'.

`-WOPT:mem_opnds=`*ON*|*OFF*

The compiler's scalar optimizer is instructed to use automated reasoning to protect all memory operands of arithmetic operations. The process attempts to incorporate memory loads as part of the operands of arithmetic operations (e.g., the compiler tries to combine a memory load and an arithmetic instruction into one instruction). The default is '`WOPT:mem_opnds=OFF`'.

`-WOPT:retype_expr=`*ON*|*OFF*

Whenever possible the compiler calculate 64-bit addresses using 32-bit arithmetic. The default is '`-WOPT:retype_expr=OFF`'.

`-WOPT:unroll=`*0*|*1*|*2*

Specifying '`WOPT:unroll`' helps regulate the compiler's scalar optimizer when unrolling of innermost loops. The available settings are:

0 Innermost loop unrolling is suppressed.

1 Instructs the compiler's scalar optimizer to unroll the innermost loops which contain `IF` statements. Selecting this setting complements the loop unrolling performed in the code generator.

2 Instructs the compiler's scalar optimizer to unroll the innermost loops which contain straight line code plus the loops containing `IF` statements. Selecting this setting duplicates the unrolling performed in the code generator (i.e. unrolling straight line code in the body of a loop).

Note '`-WOPT:unroll`' and the unrolling options in the '`-OPT`' group are mutually exclusive. The default is '`-WOPT:unroll=1`'.

`-WOPT:val=`*0|1|2*

The compiler attempts to identify expressions which compute identical runtime values. Then proceeds to adjust the code to avoid recomputing the values. Setting '`-WOPT:val`' tells the global optimizer the number of times the value-numbering optimization should be performed. The default is '`-WOPT:val=1`'.

3.9.3 Options that Control General Optimizations

The options below control general optimizations that are not associated with a specific compilation phase.

The following options control specific optimizations. They are either activated by '`-O`' options or are related to ones that are. You can use the following flags in the rare cases when "fine-tuning" of optimizations to be performed is desired.

The following options control compiler behavior regarding floating point arithmetic. These options trade off between speed and precision. All must be specifically enabled.

`-chunk=`*N* Sets the default chunk size to *N*. When the '`-apo`' and '`-mp`' options are used, loops may be parallelized with different iterations of a loop being scheduled to execute on different threads. The chunk size is the number of consecutive iterations of a parallel loop assigned to a thread each time work is scheduled for a thread. The default is the total number of iterations divided by the number of threads.

`-ffast-math`
`-fno-fast-math`

Instructs the compiler to relax ANSI/ISO or IEEE rules/specifications for math functions in order to optimize floating-point computations to improve runtime. '`-fno-fast-math`' instructs the compiler to conform to ANSI and IEEE math rules.

This option causes the preprocessor macro `__FAST_MATH__` to be defined.

Note:

'`-Ofast`' implies '`-ffast-math`'.
'`-ffast-math`' sets options '`-fno-math-errno`' and '`-OPT:IEEE_arithmetic=2`'.
'`-fno-fast-math`' sets options '`-fmath-errno`' and '`-OPT:IEEE arithmetic=1`'.

`-ffloat-store`

Do not store floating-point variables in registers, and inhibit other options that might change whether a floating point value is taken from a register or memory.

This option prevents undesirable excess precision on the computations performed in the floating-point unit, regardless of the original type (e.g., registers keep more precision than a `double` is required to have). For most programs, the excess precision does only good, but a few programs rely on the precise definition of IEEE floating point. Use '`-ffloat-store`' for such programs, after modifying them to store all pertinent intermediate computations into variables. '`-ffloat-store`' generates stores to memory of all pertinent immediate computations to be truncated to a lower precision which may generate extra stores

slowing down program execution. (See [‘-mx87-precision’], page 54, for more details). Note ‘-ffloat-store’has no effect under ‘-msse2’, the default when specifying ‘-m32’ and ‘-m64’.

-fno-math-errno
: Do not set ERRNO after calling math functions that are executed with a single instruction, e.g., sqrt. A program that relies on IEEE exceptions for math error handling may want to use this flag for speed while maintaining IEEE arithmetic compatibility. Note specifying ‘-Ofast’ implies ‘-fno-math-errno’. The default is ‘-fmath-errno’.

-funsafe-math-optimizations
-fno-unsafe-math-optimizations
: ‘-funsafe-math-optimizations’ instructs the compiler to allow optimizations for floating-point arithmetic that assume that arguments and results are valid, and may violate IEEE or ANSI standards. When used at link-time, it may include libraries or startup files that change the default floating-point unit control word or other similar optimizations. ‘-fno-unsafe-math-optimizations’ instructs the compiler to conform to ANSI and IEEE math rules. The default is ‘-fno-unsafe-math-optimizations’.

-mx87-precision=*32|64|80*
: Specifies the floating-point precision of the floating-point units calculations. The three settings available are: 32-bit, 64-bit, or 80-bit. The default is ‘-mx87-precision=80’.

-noexpopt
: Instructs the compiler to **not** optimize exponential operations.

-openmp
-mp
: Instructs the compiler to interpret OpenMP directives to explicitly parallelize specified code for multi-thread execution on shared-memory multiprocessor models. The opencc, openCC, and openf95 compilers support directives for OpenMP 2.5.

-OPT:*question=answer*
: The ‘-OPT:’ option group controls various optimizations. The ‘-OPT:’ options supersede the defaults that are based on the main optimization level.

  -OPT:alias=*model*
  : Identify which pointer aliasing model to use. The compiler will make assumptions during compilation when one or more of the following *model* is specified.

    typed
    : Assumes that two pointers of different types will **not** point to the same location in memory (i.e. the code adheres to the ANSI/ISO C standards). Note when specifying ‘-OPT:Ofast’ turns this option *ON*.

    restrict
    : Assumes that distinct pointers are pointing to distinct non-overlapping objects. This optimization is disabled by default.

disjoint Assumes that any two pointer expressions are pointing to distinct non-overlapping objects. This optimization is disabled by default.

no_f90_pointer_alias
Assumes that any two different Fortran 90 pointers are pointing to distinct non-overlapping objects. This optimization is disabled by default.

`-OPT:align_unsafe=`*ON*|*OFF*
Assumes that array parameters are aligned at 128-bit boundaries and instructs the vectorizor to aggressively vectorize the code. The vectorizor then proceeds to generate 128-bit aligned `load` and `store` instructions. Note the aligned memory accesses will execute faster than unaligned accesses, but if the assumption is faulty the aligned memory accesses will result in runtime segmentation faults. The default is '`-OPT:align_unsafe=`*OFF*'.

`-OPT:asm_memory=`*ON*|*OFF*
Assumes each inline assembly instruction has specified memory (i.e. even if it is not available). Note this switch can be used to debug suspicious inline assembly code. The default is '`-OPT:asm_memory=`*OFF*'.

`-OPT:bb=`*N*
The value *N* limits the number of instructions a basic block can contain in the code generator's program representation. A basic block is defined as the straight line sequence of instructions with no control flow. Note the larger the value *N* the greater the opportunity exists for applying optimizations at the basic block level. Compiling programs where *N* is large and that exhibit large basic blocks could increase compilation time. The default is '`-OPT:bb=`*1300*'. Select a smaller value if compilation time becomes an issue.

`-OPT:cis=`*ON*|*OFF*
`SIN`/`COS` pairs that use identical arguments are converted to a single call and both values are calculated at once. The default is '`-OPT:cis=`*ON*'.

`-OPT:cyg_instr=`*0*|*1*|*2*|*3*|*4*
Instructs the compiler to insert instrumentation calls into each function. Instrumentation calls are inserted following the function entry and just before the function returns. Example insertions:

```
void__cyg_profile_func_entry (void *func_address, void *return_address);
void__cyg_profile_func_exit (void *func_address, void *return_address);
```

Where, the first argument is the address at the start of the current function and the second argument is the return address into the caller of the current function.

'`-OPT:cyg_instr`' has five settings that control which functions are **not** instrumented:

0 Do **not** instrument any function.

1 Do **not** instrument functions the GNU front-end selects for inlining.

2 Do **not** instrument functions marked `inline` in the source.

3 Do **not** instrument functions marked `extern inline` or `always_inline`.

4 Instrument all functions and disable deletion of `extern inline` functions. Specifying this value may create linking and runtime faults.

Note options '`-finstrument-function`' and '`-OPT:cyg_instr=3`' are equivalent, See ['`-finstrument-functions`'], page 49.

For any function assigned the attribute `no_instrument_function`, instrumentation will be suppressed (e.g., do **not** instrument functions __cyg_profile_func_enter and __cyg_profile_func_exit).

`-OPT:div_split=`*`ON`*`|`*`OFF`*

Instruct the compiler to transform `x/y` into `x*(recip(y))`. Flags '`-OPT:Ofast`' or '`-OPT:IEEE_arithmetic=3`' will enable this option. Note this transformation generates fairly accurate code. The default is '`-OPT:div_split=OFF`'.

`-OPT:early_mp=`*`ON`*`|`*`OFF`*

Instructs the compiler to transform code to execute under multiple threads only before or after the loop nest optimization (LNO) phase in the compilation process. When '`-OPT:early_mp=ON`' some OpenMP programs yield better performance because LNO is allowed to generate appropriate loop transformations when working on the multi-threaded forms of loops. Note if '`-apo`' is specified the transformation of code executing multiple threads can only take place after LNO phase. In which case the '`-OPT:early_mp`' flag is ignored.

The default is '`-OPT:early_mp=OFF`'.

`-OPT:early_intrinsics=`*`ON`*`|`*`OFF`*

Generate calls to intrinsics which can be expanded to inline code early in the back-end compilation. The early inlining could expose short-vector forms in the expanded code leading to vectorization opportunities. The default is '`-OPT:early_intrinsics=OFF`'.

`-OPT:fast_bit_intrinsics=`*`ON`*`|`*`OFF`* (Fortran Only)

If '`-OPT:fast_bit_intrinsics=ON`' the check for the bit count being within range for Fortran intrinsics (e.g., `BTEST` or `ISHFT`) will be turned off. The default is '`-OPT:fast_bit_intrinsics=OFF`'.

`-OPT:fast_complex=`*`ON`*`|`*`OFF`*

Specifies fast calculations for values declared to be of the type `complex`. Fast algorithms are used for complex absolute value

and complex division. The algorithm will overflow for an operand (e.g., the divisor in the case of division) that has an absolute value that is larger than the square root of the largest representable floating-point number. Also, the algorithm will underflow for a value that is smaller than the square root of the smallest representable floating point number. Note, specifying '`-OPT:roundoff=3`' will also set '`-OPT:fast_complex=ON`'. The default is '`-OPT:fast_complex=OFF`'.

`-OPT:fast_exp=`*`ON`*`|`*`OFF`

Transforms exponentiation by integers or halves to sequences of multiplies and square roots. This option can affect roundoff, and can make these functions produce minor discontinuities at the exponents where it applies. Note specifying '`-O3`', '`-Ofast`', or '`-OPT:roundoff=1`' will enable this option. The default is '`-OPT:fast_exp=OFF`'.

`-OPT:fast_io=`*`ON`*`|`*`OFF`* (C/C++ Only)

Instruct the compiler to enable inlining of `printf()`, `fprintf()`, `sprintf()`, `scanf()`, `fscanf()`, `sscanf()`, and `printw()` for more specialized lower-level subroutines. The option invokes inlining only if the prospects are marked as intrinsic in the respective header files (i.e. `<stdio.h>` and `<curses.h>`) Note programs that use I/O functions (e.g., `printf()` or `scanf()` extensively generally have improved I/O performance when this flag is enabled. Use of this option may create substantial code expansion. The default is '`-OPT:fast_io=OFF`'.

`-OPT:fast_math=`*`ON`*`|`*`OFF`

Instructs the compiler to use the fast math functions tuned for the target processor. The fast math functions include log, exp, sin, cos, sincos, expf, and pow. Note '`-OPT:fast_math=ON`' when '`-OPT:roundoff`' is set to be equal to or greater than 2. The default is '`-OPT:fast_math=OFF`'.

`-OPT:fast_nint=`*`ON`*`|`*`OFF`

Instructs the compiler to use hardware features to implement single-precision and double-precision `NINT` and `ANINT`. Note if '`-OPT:roundoff=3`' is specified then '`-OPT:fast_nint=ON`'. The default is '`-OPT:fast_nint=OFF`'.

`-OPT:fast_sqrt=`*`ON`*`|`*`OFF`

Instructs the compiler to calculate the square root using the identity `sqrt(x)= x*rsqrt(x)` (where `rsqrt` equals the reciprocal square root operation). Fairly accurate code is generated when specifying this transformation. Note '`-OPT:fast_exp=ON`' must be specified which instructs the compiler to generate inlined instructions and **not** to call the library `pow` function before '`-OPT:fast_sqrt=ON`' can perform its transformation. '`-OPT:fast_sqrt`' has no dependencies on '`-OPT:rsqrt`' and unlike '`-OPT:rsqrt`' the transformation

request by '-OPT:fast_sqrt' does **not** generate extra instructions when implementing rsqrt. The default is '-OPT:fast_sqrt=OFF'.

-OPT:fast_stdlib=*ON*|*OFF*
: Instructs the compiler to generate calls to faster versions of standard library functions. The default is '-OPT:fast_stdlib=ON'.

-OPT:fast_trunc=*ON*|*OFF*
: Instructs the compiler to inline the single-precision and double-precision versions of the Fortran intrinsics NINT, ANINT, and AMOD. Note '-OPT:fast_trunc=ON' is specified if '-OPT:roundoff' is equal to or greater than 1. The default is '-OPT:fast_trunc=OFF'.

-OPT:fold_reassociate=*ON*|*OFF*
: When set to *ON* the compiler performs transformations which reassociate or distribute a floating-point expression. Note, specifying '-O3' or '-OPT:rounoff' to be equal to or greater than 2, forces '-OPT:fold_reassociate=ON' . The default is '-OPT:fold_reassociate=OFF'.

-OPT:fold_unsafe_relops=*ON*|*OFF*
: Instructs the compiler to fold relational operators that will transform possible integer overflow.

  Note, specifying '-O3' sets '-OPT:fold_unsafe_relops=ON'. The default is '-OPT:fold_unsafe_relops=OFF'.

-OPT:fold_unsigned_relops=*ON*|*OFF*
: Instructs the compiler to fold relational operators involving unsigned integers that may be simplified at compile time, which may cause fewer overflows to be seen at run time. The default is '-OPT:fold_unsigned_relops=ON'

-OPT:goto=*ON*|*OFF*
: Transforms GOTO into higher level structures e.g., FOR loops. Note if '-O2' is specified then '-OPT:goto=ON'. The default is '-OPT:goto=OFF'.

-OPT:IEEE_arithmetic=*1*|*2*|*3*
-OPT:IEEE_arith=*1*|*2*|*3*
-OPT:IEEE_a=*1*|*2*|*3*
: This flag regulates the level of conformance to ANSI/IEEE 754-1985 floating point roundoff and overflow. Note '-OPT:IEEE_arith' and '-OPT:IEEE_a' are valid abbreviations for the option. The levels of conformance:

  1 — Adhere to IEEE 754 accuracy. Specifying '-O0', '-O1', and '-O2' will set '-OPT:IEEE_arithmetic=1'.

  2 — Inexact results that do **not** conform to IEEE 754 may be calculated. Specifying '-O3' will set '-OPT:IEEE_arithmetic=2'.

3 All valid mathematical transformations (possibly non-IEEE standard ones) are allowed.

`-OPT:IEEE_NaN_inf=`*ON*`|`*OFF*

Instructs the compiler to conform to ANSI/IEEE 754-1985 for all operations which produce a NaN or infinity result. Note NaN and infinity are typically handled as special cases in floating-point representations of real numbers and are defined by the IEEE 754 Standards for Binary Floating-point Arithmetic.

When setting this option to *OFF*, various operations that do not produce IEEE-754 results. For example, `x/x` is set to the value 1 without performing a divide operation and `x=x` is set to TRUE without executing a test operation. '`-OPT:IEEE_NaN_inf=OFF`' also specifies multiple optimizations that increase performance. The default is '`-OPT:IEEE_NaN_inf=OFF.`'

`-OPT:inline_intrinsics=`*ON*`|`*OFF* (Fortran Only)

'`-OPT:inline_intrinsics=OFF`' transforms all Fortran intrinsics that have a library function into a call to that function. The default is '`-OPT:inline_intrinsics=ON`'.

`-OPT:keep_ext=`*ON*`|`*OFF*

Instructs the compiler to preserve external symbolic information. The default is '`-OPT:keep_ext=OFF`'.

`-OPT:malloc_algorithm=`*0*`|`*1*

`-OPT:malloc_alg=`*0*`|`*1*

To improve runtime speed the compiler will select an optimal malloc algorithm. To enable the selected algorithm, setup code is included in the C/C++ and Fortran `main` function. The default is '`-OPT:malloc_algorithm=0`'

`-OPT:Ofast`

Maximizes performance for a given platform using the selected optimizations. '`-OPT:Ofast`' specifies four optimizations: '`-OPT:ro=2`', '`-OPT:Olimit=0`', '`-OPT:div_split=ON`', and '`-OPT:alias=typed`'. Note the specified optimizations are ordinarily safe but floating point accuracy due to transformations may be diminished.

`-OPT:Olimit=`*N*

Controls the size of procedures to be optimized. Procedures above the specified cutoff limit, *N*, are not optimized. *N*=0 means "infinite Olimit," which causes all procedures to be optimized with **no** consideration regarding compilation times. Note if '`-OPT:Ofast`' is enabled then '`-OPT:Olimit=0`' or when '`-O3`' is enabled '`-OPT:Olimit=9000`'. The default is '`-OPT:Olimit=6000`'.

`-OPT:pad_common=`*ON*`|`*OFF*

Instructs the compiler to reorganize common blocks in order to optimize the cache behavior of accesses to members of the common

block. To achieve the optimum, additional padding between members of blocks and/or dividing a common block into a group of blocks may be required. Note:

- do **not** specify this option unless the common block definitions (i.e. including `EQUIVALENCE`) are consistent among all source codes making up a program.
- do **not** specify '`-OPT:pad_common=ON`' if common blocks are initialized with `DATA` statements
- Specifying '`-OPT:pad_common=ON`' implies that all source codes in the program conform to this transformation.

The default is '`-OPT:pad_common=OFF`'

`-OPT:recip=`*`ON`*`|`*`OFF`*

Instructs the compiler to use the reciprocal instruction for `1.0/y` This may change the accuracy of the results. For example, `X = 1./Y` generates the reciprocal instruction instead of a divide instruction. The default is '`-OPT:recip=OFF`'.

`-OPT:reorg_common=`*`ON`*`|`*`OFF`*

Instructs the compiler to reorganize common block in order to optimize the cache behavior of accesses to members of the common block. If the compiler determines it is safe, it will proceed with the common block reorganization. Note defaults are:

`-OPT:reorg_common=ON`

When '`-O3`' is specified, plus all source files which reference the common block are compiler with '`-O3`'.

`_OPT:reorg_common=OFF`

'`-O2`' or below is specified when compiling the file that contains the common block.

`-OPT:roundoff=`*`0`*`|`*`1`*`|`*`2`*`|`*`3`*
`-OPT:ro=`*`0`*`|`*`1`*`|`*`2`*`|`*`3`*

'`-OPT:roundoff`' specifies acceptable levels of divergence for both accuracy and overflow/underflow behavior of floating-point results relative to the source language rules. The roundoff value has a value in the range *0* to *3* and are described in the following table:

0 Do **no** transformations which could affect floating-point results. The is the default for optimization levels '`-O0`', '`-O1`', and '`-O2`'.

1 Allow all transformations which have a limited affect on floating-point results. For roundoff, limited is defined as only the last bit or two of the mantissa is affected. For overflow or underflow, limited is defined as intermediate results of the transformed calculation may overflow or underflow within a factor of two where

the original expression may have overflowed or underflowed. Note that effects may be less limited when compounded by multiple transformations. This is the default when '`-O3`' is specified.

2 Specifies transformations with extensive effects on floating-point results. For example, allow associative rearrangement (i.e. even across loop iterations) and the distribution of multiplication over addition or subtraction. Do **not** specify transformations known to cause:

- cumulative roundoff errors.
- overflow/underflow of operands in a large range of valid floating-point values.

This is the default when specifying '`-OPT:Ofast`'

3 Specify any mathematically valid transformation of floating-point expressions. For example, floating point induction variables in loops are permitted (even if known to cause cumulative roundoff errors). Also permitted are fast algorithms for complex absolute value and divide (which will overflow/underflow for operands beyond the square root of the representable extremes).

`-OPT:rsqrt=`*0*`|`*1*`|`*2*

Instructs the compiler to use the reciprocal square root instruction when calculating the square root. This transformation may vary the accuracy slightly.

0 Restrain from using the reciprocal square root instruction

1 Use the reciprocal square root instruction followed by operations that will improve the accuracy of the results.

2 Use the reciprocal square root instruction **without** improving the result accuracy.

Note specifying '`-OPT:roudoff=2`' or '`-OPT:roundoff=3`' will set '`-OPT:rsqrt=1`'. The default is '`-OPT:rsqrt=0`'.

`-OPT:space=`*ON*`|`*OFF*

Instructs the compiler to consider code size as a higher priority then execution time when performing optimization. Note '`-Os`' sets this option to equal *ON*. The default is '`-OPT:space=OFF`'.

`-OPT:speculate=`*ON*`|`*OFF*

Instructs the compiler to make an effort to eliminate branches at the expense of increasing the computations. Whenever possible, the compiler transforms short-circuiting conditionals

to comparable non-short-circuited structures. The default is '`-OPT:speculate=OFF`'.

`-OPT:transform_to_memlib=`*`ON|OFF`*
: Instructs the compiler to perform loop transformations by using library function calls to `memcpy` or `memset`. The default is '`-OPT:transform_to_memlib=ON`'.

`-OPT:treeheight=`*`ON|OFF`*
: Instructs the compiler to perform a global reassociation of expressions which reduces the tree height of the expressions. The reassociation process determines the optimum order of combining terms in a sum so as to produce loop invariant constant subcomputations or to refine common subexpressions among several essential computations. The default is '`-OPT:treeheight=OFF`'.

`-OPT:unroll_analysis=`*`ON|OFF`*
: Instructs the compiler to disregard both '`-OPT:unroll_times_max`' and '`-OPT:unroll_size`'. The compiler then proceeds to perform a global analysis of the loop constructs to determine the optimum loop unrolling parameters. As a result of the analysis minimal loop unrolling which reduces code size may occur, permitting a faster compilation. Note specifying '`-OPT:unroll_analysis=ON`' negates the effects of '`-OPT:unroll_times_max`' and '`-OPT:unroll_size`' causing loops to be unrolled by less than the upper limit. The default is '`-OPT:unroll_analysis=ON`'.

`-OPT:unroll_level=`*`1|2`*
: Controls the level at which the compiler will perform unrolling optimizations. When '`-OPT:unroll_level=2`' the compiler is instructed to aggressively unroll loops in the presence of control flow. The default is '`-OPT:unroll_level=1`'.

`-OPT:unroll_times_max=`*`N`*
: Instructs the compiler to limit the unrolling of inner loops to the value specified by *N*. The default is '`-OPT:unroll_times_max=4`'.

`-OPT:unroll_size=`*`N`*
: Instructs the compiler to limit the number of instructions produced when unrolling inner loops. When N=0 the ceiling is disregarded. Note specifying '`-O3`' sets '`-OPT:unroll_size=128`'. The default is '`-OPT:unroll_size=40`'.

`-OPT:wrap_around_unsafe_opt=`*`ON|OFF`*
: '`-OPT:wrap_around_unsafe_opt=OFF`' instructs the compiler **not** to perform induction variable replacement and linear function test replacement. Both of these transformations are enabled when specifying '`-O3`'. The option is provided as a diagnostic tool. Note specifying '`-O0`' sets '`-OPT:wrap_around_unsafe_opt=OFF`'. When '`-OPT:wrap_around_unsafe_opt=OFF`' the performance may be degraded.

3.9.4 Options that Control Interprocedural Optimizations

Inline expansion, or inlining, is a compiler optimization that replaces a function call with the body of the function. This optimization may improve time and space usage at runtime, at the possible cost of increasing the final code size.

`-fimplicit-inline-templates` (C++ Only)
`-fno-implicit-inline-templates` (C++ Only)
: '`-fimplicit-inline-templates`' instructs the compiler to emit code for implicit instantiations of inline templates. '`-fno-implicit-inline-templates`' will never emit code for implicit instantiations of inline templates. The default is '`-fno-implicit-inline-templates`'.

`-fimplicit-templates` (C++ Only)
`-fno-implicit-templates` (C++ Only)
: '`-fimplicit-templates`' instructs the compiler to emit code for non-inline templates which are instantiated implicity. '`-fno-implicit-templates`' will never emit code for non-inline templates which are instantiated implicitly (i.e. by use); it will only emit code for explicit instantiations.

`-finline`
`-fno-inline`
`-inline`
`-INLINE`
`-noinline`
: Options '`-finline`','`-INLINE`' and '`-inline`' instruct the compiler to perform inline processing (i.e. expansion of inline functions). If optimizations are **not** being performed then function inlining is suppressed. '`-fno-inline`' and '`-noinline`' disable inlining and don't pay attention to the `inline` keyword. Note when performing interprocedural analysis (IPA) then '`-IPA:inline=OFF`' must be specified when disabling inlining.

`-finline-functions` (C/C++ Only)
`-fno-inline-functions` (C/C++ Only)
: '`-finline-functions`' automatically integrates simple functions (i.e. the callees) into the callers. The compiler heuristically decides which functions are simple enough to be worth integrating in this way. If all calls to a given function are integrated, and the function is declared `static`, then normally assembler code is not generated for the function. Specifying '`-fno-inline-functions`' will disable the automatic integration of simple functions. Note this option is enabled at optimization level 3, '`-O3`'. The default is '`-fno-inline-function`'.

`-fkeep-inline-functions` (C/C++ Only)
: Instructs the compiler to generate code for functions even if they are fully inlined. In C, emit code for `static` functions that are declared `inline` into the object file, even if the function has been inlined into all of its callers. This switch does not affect functions using the `extern inline` extension in C. In C++, emit any and all inline functions into the object file.

`-INLINE:`*question=answer*

This option group transforms function calls by use of inlining. If inlining directives are inserted in the source code then the '`-INLINE`' option must be specified in order for those directives to be recognized. Note when specifying the '`-INLINE`' option the program may not always compile successfully, with the exception of '`-INLINE:=OFF`' which suppresses the invocation of the lightweight inliner.

`-INLINE:all`

Instructs the compiler to perform all possible inlining. Note since inlining increases the code size, this option should be specified with some discretion (e.g., use only if program is small).

`-INLINE:aggressive=`*ON*`|`*OFF*

Instructs the compiler to be very aggressive when performing inlining. The default is '`-INLINE:aggressive=OFF`'.

`-INLINE:list=`*ON*`|`*OFF*

Instructs the compiler to emit a list of inlining transformations as they occur to '`stderr`'. The emitted comments outline which functions are inlined, which functions are not inlined, and why. The default is '`-INLINE:list=OFF`'.

`-INLINE:must=`*name1*`[,`*name2*`,...]`
`-INLINE:never=`*name1*`[,`*name2*`,...]`

Functions can be tagged for inlining by specifying the function name when using the '`-INLINE:must`' option or suppressed by using the option '`-INLINE:never`'. Note when using this option in C++, use the C++ mangled name for the function.

`-INLINE:none`

Disables automatic inlining specified by the interprocedural analysis group option ('`-IPA:`'). Note inlining specified by a command-line option or implied by the language are still performed. The default is automatic inlining is turned ON.

`-INLINE:preempt=`*ON*`|`*OFF*

Inline functions labeled preemptible in the lightweight inliner. Preempt inlining prevents alternate definitions of a function, in another dynamic shared object (DSO), from preempting the definition of the function being inlined. The default is '`-INLINE:preempt=OFF`'.

`-ipa`
`-IPA`
`-IPA:` Instructs the compiler to invoke interprocedural analysis. These options are identical. The default settings for the '`-IPA:`*question=answer*' option group are invoked. Specifying '`-ipa`' is equivalent to '`-IPA`' and '`-IPA:`' with no suboptions.

`-IPA:`*question=answer*

The '`-IPA:`' option group commands the compiler to perform interprocedural analysis and optimization. Interprocedural analysis and optimization can

include: inlining, common block array padding, constant propagation, dead function elimination, alias analysis, and others. If the option group defaults are acceptable for compilation then '`-IPA:`' can be specified without specifying arguments. Note if compiling and linking are performed in separate steps then '`-IPA:`' must be specified for compilation step and for the link step. An error will occur if '`-IPA:`' is specified for the compile step and **not** for the link step.

`-IPA:addressing=`*ON*|*OFF*
: Enabling this option invokes the analysis of address operator usage. Setting '`-IPA:alias=ON`' is a prerequisite for this option. The default is '`-IPA:addressing=OFF`'.

`-IPA:aggr_cprop=`*ON*|*OFF*
: The compiler is instructed to perform aggressive interprocedural constant propagation. Interprocedural constant propagation is a process which replaces formal parameters by their corresponding constant values. This process strives to prevent passing constant parameters. The conventional interprocedural constant propagation ('`-IPA:cprop=ON`') is performed by default. The default is '`-IPA:aggr_cprop=OFF`'.

`-IPA:alias=`*ON*|*OFF*
: The compiler is instructed to perform `ALIAS`, `MOD`, and `REF` analysis. '`-IPA:alias=ON`' specifies an interprocedural analysis that provides results not affected by control flow in procedures. The default is '`-IPA:alias=ON`'.

`-IPA:callee_limit=`*N*
: The compiler is instructed **not** to allow inlining of functions with a compiler-evaluated internal code size larger than the limit set by *N*. The default is '`-IPA:callee_limit=500`'.

`-IPA:cgi=`*ON*|*OFF*
: The compiler is instructed to identify constant global variables. The compiler tags all non-scalar global variables which are **not** modified as constants, and never passes their constant values to all programs. The default is '`-IPA:cgi=ON`'.

`-IPA:clone_list=`*ON*|*OFF*
: The compiler is instructed to use the interprocedural analysis function cloner to list all cloning actions performed by the compiler to '`stderr`'. The default is '`-IPA:clone_list=OFF`'.

`-IPA:common_pad_size=`*N*
: The compiler is instructed to pad common block array dimensions by the value *N*. By default, the compiler automatically chooses the amount of padding to improve cache behavior for common block array accesses. The default is '`-IPA:common_pad_size=0`'.

`-IPA:cprop=`*ON*|*OFF*
: The compiler is instructed to perform interprocedural constant propagation. Interprocedural constant propagation is a process

which replaces formal parameters that always have a specific constant values. The default is '`-IPA:cprop=ON`'.

`-IPA:ctype=`*`ON`*`|`*`OFF`

Instructs the compiler to generate accelerated versions of the `<ctype.h>` header macros (e.g., `isalpha()`, `isascii()`,...). Note using this option when compiling multithreaded code and in all locales (i.e. other than the 7-bit ASCII or C-language locale) may generate unstable executables. The default is '`-IPA:ctype=OFF`'

`-IPA:depth=`*`N`*

'`-IPA:depth=`*`N`*' is equivalent to '`-IPA:maxdepth=`*`N`*'.

`-IPA:dfe=`*`ON`*`|`*`OFF`

The compiler is instructed to perform "dead function elimination" (dfe). The compiler removes all subprograms/functions which are never called from the program. Note when a function call is replaced by inlining the function everywhere in the program then the original function becomes a "dead function" and is eliminated. The default is '`-IPA:dfe=ON`'.

`-IPA:dve=`*`ON`*`|`*`OFF`

The compiler is instructed to perform "dead variable elimination" (dve). The compiler removes all variables which are never referenced by the program. The default is '`-IPA:dve=ON`'.

`-IPA:echo=`*`ON`*`|`*`OFF`

Instructs the compiler to echo the compiler and final link commands which are invoked from IPA to '`stderr`'. '`-IPA:echo=ON`' can assist in monitoring the progress of a large class program build. The default is '`-IPA:echo=OFF`'.

`-IPA:field_reorder=`*`ON`*`|`*`OFF`

The compiler is instructed to reorder fields in large structures based on the fields reference patterns in feedback compilation. The field reordering minimizes data cache misses. The default is '`-IPA:field_reorder=OFF`'.

`-IPA:forcedepth=`*`N`*

When attempting to inline functions, the compiler is instructed to limit the depth in the callgraph to *N*. Note depth *0* refers to functions making **no** calls, depth *1* are those calling only depth *0* functions, and so on. By default '`-IPA:forcedepth`' is ignored and the heuristic limits on inlining are in effect.

`-IPA:ignore_lang=`*`ON`*`|`*`OFF`

When performing inlining, the compiler is instructed to ignore language boundaries (i.e. inlining across the Fortran language to C/C++ language boundary). Note the compiler may **not** always be cognizant of the proper language semantics and this optimization may have effects at runtime producing unreliable or improper conclusions. The default is '`-IPA:ignore_lang=OFF`'.

`-IPA:inline=`*`ON`*`|`*`OFF`*

The compiler is instructed to perform inter-file subprogram inlining during main interprocedural analysis processing. The default is '`-IPA:inline=ON`'.

`-IPA:keeplight=`*`ON`*`|`*`OFF`*

Specifying the flag '`-keeplight`' instructs the compiler to save space. The default is '`-IPA:keeplight=OFF`'.

`-IPA:linear=`*`ON`*`|`*`OFF`*

The compiler is instructed to perform linearization of array references. The compiler transforms a multi-dimensional array to a linear array (i.e. single dimensional)which is mapped to the same memory block. The compiler attempts to map formal array parameters to the shape of the actual parameter when inlining Fortran subroutines. This mapping process may not always be successful, therefore when the compiler is unable to map the parameter it linearizes the array reference. Note the compiler will **not** attempt to inline such callsites due to possible performance problems. The default is '`-IPA:linear=OFF`'.

`-IPA:map_limit=`*`N`*

The interprocedural analysis process uses *N* as a threshold to trigger invoking '`-IPA:sp_partition`'. When the size of the input files mapped exceeds the limit, *N* bytes, the compiler enables '`-IPA:sp_partition`'. The default is '`-IPA:map_limit`' is ignored.

`-IPA:maxdepth=`*`N`*

The compiler is instructed to **not** attempt to inline functions at a depth in the callgraph which exceeds *N*. Note depth *0* refers to functions making **no** calls, depth *1* are those calling only depth *0* functions, and so on. By default '`-IPA:maxdepth`' is ignored and the heuristic limits on inlining are in effect.

`-IPA:max_jobs=`*`N`*

The compiler is instructed to limit the number of compilation running at once to *N*. After interprocedural analysis is performed the compiler is invoked with '`-IPA:max_job`' set to the maximum level of parallelism. *N* can be set to:

0 The maximum level of parallelism is limited to the greatest number of:

- CPUs (or processor sockets),
- processor cores,
- or hyperthreading units

in the system.

1 Suppress parallelization during compilation.

>=*2* Sets the desired level of parallelism

The default is '`-IPA:max_jobs=1`'.

`-IPA:min_hotness=`*N*

The compiler is instructed **not** to inline a function to a call site (i.e. caller) unless the callee is invoked more than *N* times. The compiler examines the interprocedural feedback to determine if the threshold set by *N* is surpassed by a call site to a procedure and then proceeds to inline the procedure if the limit is exceeded. The default is '`-IPA:min_hotness=10`'.

`-IPA:multi_clone=`*N*

Specifies to the compiler the maximum number of clones of a single procedure the compiler can create. Setting *N* to a large number promotes more opportunities for interprocedural optimizations. Note significant increase in code size may occur if aggressive procedural cloning is performed. The default is '`-IPA:multi_clone=0`'.

`-IPA:node_bloat=`*N*

When the compiler is invoked with option '`-IPA:multi_clone`', '`-IPA:node_bloat`' can be specified to gage the code size growth due to procedural cloning. *N* specifies the maximum percentage of code size growth (i.e. relative to the original code size) due to the total number of procedures cloned. The default is '`-IPA:node_bloat=100`'.

`-IPA:plimit=`*N*

The compiler is instructed to halt inlining within a program once the intermediate representation indicates that the code size of the program has surpassed the limit set by *N*. The default is '`-IPA:plimit=2500`'.

`-IPA:pu_reorder=`*0*|*1*|*2*

The compiler is instructed to examine compilation feedback for invocation patterns to determine the process of reordering the layout of program procedures in order to minimize instruction cache misses. Possible settings are:

0 Suppress reordering of program procedures.

1 Use the frequent occurrence of procedure invocation to determine reordering.

2 Use the relationship between caller and callee to determine reordering.

The default is '`-IPA:pu_reorder=1`' for C++ programs and '`-IPA:pu_reorder=0`' for non-C++ programs.

`-IPA:relopt=`*ON*|*OFF*

The compiler is instructed to build objects under the presumption that the compiled objects will be linked into a call-shared

executable. The default is '`-IPA:relopt=OFF`'. When invoked as '`-IPA:relopt=ON`', optimizations based on position-dependent code (non-PIC) will be performed on the compiled objects. Note '`-IPA:relopt`' is similar to invoking the compiler using '`-O`' and '`-c`'.

`-IPA:small_pu=`*N*

The compiler is instructed **not** to restrict a procedure from inlining with a code size smaller than *N* when invoking the '`-IPA:plimit`' flag. The default is '`-IPA:small_pu=30`'.

`-IPA:sp_partition=`*ON*|*OFF*

The compiler is instructed to use partitioning when building huge programs. Note partitioning is normally performed internal to the IPA. This option is invoked to conserve disk and memory space. The default is '`-IPA:sp_partition=OFF`'

`-IPA:space=`*N*

The compiler is instructed to perform inlining until the program code size expands by percentage specified by *N*. Therefore, to limit program code size growth to ~20%, due to inlining, specify '`-IPA:space=20`'. The default is '`-IPA:space=`'infinity.

`-IPA:specfile=`*filename*

When invoking the compiler with option '`IPA:specfile`', the user indicates that a file, *filename*, containing additional options exist. The specified file must contain zero or more lines with the additional options in the proper command line syntax. Note the nesting of '`-IPA:specfile`' is **not** permitted.

`-IPA:use_intrinsic=`*ON*|*OFF*

The compiler is instructed to load the intrinsic version of standard library functions. Note '`-IPA:use_intrinsic`' may promote the inlining of the `malloc` library function. This option improves small object allocations. The default is '`-IPA:use_intrinsic=OFF`'.

3.9.5 Options that Control Loop Nest Optimizations

`-LNO:`*question=answer*

This option group commands the compiler loop nest optimizer to perform nested loop analysis and transformations. Note an optimization level of '`-O3`' or higher must be specified in order to enable the '`-LNO:`'options.

To verify the LNO options that were invoked during compilation use the option '`-LIST:all_options=ON`'.

`-LNO:apo_use_feedback=`*ON*|*OFF*

Instructs the auto-parallelizer to use the feedback data of the loops in deciding if each loop should be parallelized. This option will only be invoked if '`-apo`' under feedback-directed compilation has been enabled. The compiler creates a serial and parallel version of

a parallelized loop and if the loop trip count is small the serial version is used during execution. When '`-LNO:apo_use_feedback=ON`' and the feedback data validates that the loop trip count is small the auto-parallelizer will **not** create the parallel version (i.e. optimizing the runtime by eliminating the conditional code required to determine the use of the serial or parallel version). The default is '`-LNO:apo_use_feedback=OFF`'.

`-LNO:build_scalar_reductions=`*`ON|OFF`*
The compiler is instructed to build scalar reductions before performing any loop transformation analysis and implementing any loop reduction transformations. Note when '`-OPT:roundoff=2`' or greater is specified then this flag is repetitious. The default is '`-LNO:build_scalar_reductions=OFF`'.

`-LNO:blocking=`*`ON|OFF`*
Instructs the compiler to perform cache blocking transformation. The default is '`-LNO:blocking=ON`'

`-LNO:blocking_size=`*`N`*
Instructs the compiler to use the specified block size, *N*, when performing all blocking. Note *N* represents the number of iterations and must be a positive integer.

`-LNO:fission=`*`0|1|2`*
Instructs the compiler to perform loop fission. This option can be set to:

0	Suppress loop fission.
1	The compiler performs normal loop fission as necessary.
2	The compiler performs loop fission prior to loop fusion.

Note loop fusion is usually applied before loop fission, therefore if '`-LNO:fission=ON`' and '`-LNO:fusion=ON`' when the compiler is invoked a reverse effect may be induced. To counter this effect specify '`-LNO:fission=2`' to instruct the compiler to perform loop fission prior to loop fusion. The default is '`-LNO:fission=0`'.

`-LNO:full_unroll=`*`N`*
`-LNO:fu=`*`N`*
The compiler is instructed to fully unroll loops after examining the loop, within the loop nest optimizer, to determine if the loop can be fully unrolled in *N* or less iterations. Argument '`fu=`*`N`*' specifies the maximum number of unrolls that can be performed to fully unroll the loop. Note setting *N*=0 suppresses full unrolling of loops inside the loop nest optimizer. '`-LNO:fu`' is the abbreviated form for '`-LNO:full_unroll`'. The default is '`-LNO:full_unroll=5`'

`-LNO:full_unroll_size=`*`N`*
The compiler is instructed to fully unroll loops after examining the loop, within the loop nest optimizer, to determine if the unrolled

loop size is less than or equal to *N*. Specify *N* as an integer between 0 and 10000. Note limits set by '`-LNO:full_unroll=`*N*' and '`-LNO:full_unroll_size=`*N*' must be satisfied before the loop is fully unrolled. The default is '`-LNO:full_unroll_size=2000`'

`-LNO:full_unroll_outer=`*ON*`|`*OFF*
The compiler is instructed to "fully" unroll loops that do **not** include inner loops or is included by an outer loop. Note limits set by both '`-LNO:full_unroll=`*N*' and '`-LNO:full_unroll_size=`*N*' must be satisfied before the loop is fully unrolled. The default is '`-LNO:full_unroll_outer=OFF`'

`-LNO:fusion=`*0*`|`*1*`|`*2*
The compiler is instructed to perform loop fusion. The flag can be set to:

0 Suppress loop fusion.

1 The compiler performs traditional loop fusion

2 The compiler performs aggressive loop fusion

The default is '`-LNO:fusion=1`'.

`-LNO:fusion_peeling_limit=`*N*
The compiler is instructed to implement loop peeling during loop fusion. *N* sets the limit on the number of loop peeling iterations. Note *N* must be greater than or equal to zero. The default is '`-LNO:fusion_peeling_limit=5`'.

`-LNO:gather_scatter=`*0*`|`*1*`|`*2*
Instructs the compiler to perform gather-scatter optimizations. The flag can be set to:

0 Suppress gather-scatter optimizations.

1 The compiler performs gather-scatter optimizations to non-nested `IF` statements.

2 The compiler performs multi-level gather-scatter optimizations.

The default is '`-LNO:gather_scatter=1`'.

`-LNO:hoistif=`*ON*`|`*OFF*
The compiler is instructed to hoist `IF` statements which reside inside inner loops. This optimization will eliminate redundant loops. The default is '`-LNO:hoistif=ON`'.

`-LNO:ignore_feedback=`*ON*`|`*OFF*
The compiler is instructed to ignore feedback information generated by loop annotations during loop nest optimizations. The default is '`-LNO:ignore_feedback=OFF`'.

`-LNO:ignore_pragmas=`*ON*`|`*OFF*
: Instructs the compiler to use command-line options instead of the corresponding directives in the source code. The default is '`-LNO:ignore_pragmas=OFF`'.

`-LNO:local_pad_size=`*N*
: The compiler is instructed to pad local array dimensions by the amount specified in *N*. The default is to instruct the compiler to choose the padding required to optimize cache behavior for local array access.

`-LNO:minvariant=`*ON*`|`*OFF*
`-LNO:minvar=`*ON*`|`*OFF*
: The compiler is instructed to move loop-invariant expressions outside of loops. '`-LNO:minvar`' is the abbreviated form for '`-LNO:minvariant`'. The default is '`-LNO:minvariant=ON`'

`-LNO:non_blocking_loads=`*ON*`|`*OFF* (C/C++ Only)
: Instructs the compiler that the target processor blocks on loads. The default is specified by the host processor in use.

`-LNO:oinvar=`*ON*`|`*OFF*
: The compiler is instructed to perform outer loop invariant hoisting. The default is '`-LNO:oinvar=ON`'

`-LNO:opt=`*0*`|`*1*
: Instructs the compiler at which level to perform loop nest optimizations. The flag can be set to:

  0 The compiler is restricted by suppress nearly all loop nest optimizations.

  1 The compiler performs full loop nest optimizations.

  The default is '`-LNO:opt=1`'

`-LNO:ou_prod_max=`*N*
: The compiler is instructed to unroll several outer loops of a given loop nest. The product is limited to *N* outer loops. Note specify *N* as a positive number. The default is '`-LNO:ou_prod_max=16`'.

`-LNO:outer=`*ON*`|`*OFF*
: The compiler is instructed to perform outer loop fusion. The default is '`-LNO:outer=ON`'.

`-LNO:outer_unroll_max=`*N*
`-LNO:ou_max=`*N*
: The compiler is instructed to perform the unrolling of outer loops of a loop nest. The unrolling of the outer loop is limited to *N* unrolls per outer loop. '`-LNO:ou_max`' is the abbreviated form for '`-LNO:outer_unroll_max`'. The default is '`-LNO:outer_unroll_max=5`'.

`-LNO:parallel_overhead=N`
: Specifying the auto-parallelizing option, '`-apo`', instructs the compiler to generate a serial and parallel instance of a loop. Specifying '`-LNO:parallel_overhead`' in conjunction with '`-apo`' instructs the compiler to estimate the overhead involved when invoking the parallel instance of the loop taking into account the number of processors and loop iterations. The loop nest optimizer then uses *N* to determine if the overhead exceeds the performance benefit during execution time. Note using this flag with auto-parallelizer can be used to tune parallel performance across various platforms and programs. The default is '`-LNO:parallel_overhead=4096`'.

`-LNO:prefetch=0|1|2|3`
: Instructs the compiler to perform prefetching optimizations at a specified level. The flag can be set to:

  0 Instructs the compiler to suppress prefetching.

  1 The compiler is instructed to allow prefetching only for arrays that are always referenced in every loop iteration.

  2 The compiler is instructed to implement prefetching disregarding the restrictions in the above setting.

  3 The compiler is instructed to implement aggressive prefetching.

  The default is '`-LNO:prefetch=2`'.

`-LNO:prefetch_ahead=N`
: The compiler is instructed to prefetch ahead *N* cache line(s). The default is '`-LNO:prefetch_ahead=2`'.

`-LNO:prefetch_verbose=ON|OFF`
: Instructs the compiler to print verbose prefetch information to '`stdout`'. The default is '`-LNO:prefetch_verbose=OFF`'.

`-LNO:processors=N`
: '`-LNO:processors`' is used in conjunction with '`-apo`' and instructs the compiler to assume that the generated code will be executed on a given number of processors installed on the target system. Specifying this flag assists in decreasing the computation time during execution when determining which loop instance to execute (i.e. serial or parallel). See ['`-LNO:parallel_overhead flag`'], page 73. Setting *N*=0 indicates the code should be compiled for a unknown number of processors and should be used when compiling programs that will be executed on platforms with different number of processors. Note the parallelized code will **not** perform optimally if *N* is set to a number that is different from the number of processors available in the target system. The default is '`-LNO:processors=0`'.

`-LNO:sclrze=`*`ON`*`|`*`OFF`*

The compiler is instructed to substitute a scalar variable for an array. The default is '`-LNO:sclrze=ON`'.

`-LNO:simd=`*`0`*`|`*`1`*`|`*`2`*

The compiler is instructed to use single instruction multiple data (SIMD) instructions, supported by the target processor, when vectorizing the inner loop. The flag can be set to:

0 The compiler is instructed to suppress vectorization.

1 The compiler is instructed to vectorize only if there is **no** performance degradation due to sub-optimal alignment and does **not** induce floating-point operation inaccuracies.

2 Instructs the compiler to aggressively vectorize with no constraints in place.

The default is '`LNO:simd=1`'.

`-LNO:simd_reduction=`*`ON`*`|`*`OFF`*

The compiler is instructed to vectorize reduction loops. The default is '`-LNO:simd_reduction=ON`'.

`-LNO:simd_verbose=`*`ON`*`|`*`OFF`*

Instructs the compiler to print verbose vectorizer information to '`stdout`'. The default is '`-LNO:simd_verbose=OFF`'.

`-LNO:svr_phase1=`*`ON`*`|`*`OFF`*

Instructs the compiler to implement the scalar variable naming phase prior to the first phase of the loop nest optimizer. The default is '`-LNO:svr_phase1=ON`'.

`-LNO:trip_count_assumed_when_unknown=`*`N`*
`-LNO:trip_count=`*`N`*

The compiler is instructed to use the value *N* as a presumed loop trip-count if at compile time a loop trip-count is **not** known. The loop trip-count, *N*, is used for loop transformations and prefetch optimizations and must be a positive integer. Note '`-LNO:trip_count`' is the abbreviated form. The default is '`-LNO:trip_count=1000`'.

`-LNO:vintr=`*`0`*`|`*`1`*`|`*`2`*

The compiler is instructed to use vector intrinsic functions when vectorizing loops. Where a vector function is called once to compute a math intrinsic for the entire vector. The flag can be set to:

0 Instructs the compiler to suppress vector intrinsic function optimizations.

1 The compiler is instructed to perform normal vector intrinsic function optimizations.

2 The compiler is instructed to aggressively implement all vector intrinsic function optimizations. Note specifying this option could cause some of the vector intrinsic functions to produce floating-point accuracy errors.

The default is '`-LNO:vintr=1`'.

`-LNO:vintr_verbose=`*`ON`*`|`*`OFF`*
Instructs the compiler to print verbose vector intrinsic optimization status to '`stdout`'. The status report will list loops that are vectorized using vector intrinsic functions. The default is '`-LNO:vintr_verbose=OFF`'.

The following **Loop Transformation Suboptions** allow the user to control cache blocking, loop unrolling, and loop interchange.

`-LNO:interchange=`*`ON`*`|`*`OFF`*
The compiler is instructed to perform loop interchange optimizations. The default is '`-LNO:interchange=ON`'.

`-LNO:unswitch=`*`ON`*`|`*`OFF`*
The compiler is instructed to perform simple loop unswitching transformations. The default is '`-LNO:unswitch=ON`'.

`-LNO:unswitch_verbose=`*`ON`*`|`*`OFF`*
Instructs the compiler to print verbose loop unswitching report to '`stdout`'. The default is '`-LNO:unswitch_verbose=OFF`'.

`-LNO:outer_unroll=`*`N`*
`-LNO:ou=`*`N`*
The compiler is instructed to unroll all outer loops by *N* iterations, i.e. when valid. The loop unrolling process is performed for *N* iterations or is **not** performed at all. '`-LNO:ou`' is the abbreviated form for '`-LNO:outer_unroll`'. *N* must be a positive integer.

`-LNO:outer_unroll_deep=`*`ON`*`|`*`OFF`*
`-LNO:ou_deep=`*`ON`*`|`*`OFF`*
The compiler is instructed to unroll the outer wind down loops which is a result of unrolling outer loops further-out. This transformation is valid for loops nested three or more deep. Note this optimization generates faster runtime code, but increase code size. Option '`-LNO:ou_deep`' is the abbreviated form. The default is '`-LNO:outer_unroll_deep=ON`'.

`-LNO:outer_unroll_further=`*`N`*
`-LNO:ou_further=`*`N`*
The compiler is instructed to perform outer loop unrolling on wind down loops. The value *N* sets a limit on the number of iterations for unrolling wind down loops. Set *N* to a positive integer. Note: Specifying '`-LNO:outer_unroll_further=999999`' suppress unrolling. Specifying '`-LNO:outer_unroll_further=3`' sets a rational limit to unrolling.

Option '`-LNO:ou_further`' is the abbreviated form. The default is '`ou_further=6`'.

`-LNO:outer_unroll_max=`*N*
`-LNO:ou_max=`*N*
: The compiler is instructed to unroll a limit of *N* copies-per-loop. Option '`-LNO:ou_max`' is the abbreviated form.

`-LNO:pwr2=`*ON*`|`*OFF* (C/C++ Only)
: When '`-LNO:pwr2=OFF`' the compiler is instructed to disregard the leading dimension. The default is '`-LNO:pwr2=ON`'.

The following **LNO Target Cache Memory Suboptions** allows the user to specify the target cache/memory system. The suboption arguments are numbered starting with the cache closest to the processor and progresses outward.

`-LNO:assoc1=`*N*`, assoc2=`*N*`, assoc3=`*N*`, assoc4=`*N*
: Instructs the compiler to set cache associativity. Setting *N* to a large number, e.g. 128, specifies a fully associative cache (e.g., main memory). Note specify *N* as a positive integer. Setting *N*=0 specifies that **no** cache exist at that level.

`-LNO:cmpl=`*N*`, cmp2=`*N*`, cmp3=`*N*`, cmp4=`*N*`, dmp1=`*N*`, dmp2=`*N*`, dmp3=`*N*`, dmp4=`*N*
: The compiler is instructed to set the time to process a clean miss (e.g. cmp*x*=*N*) or a dirty miss (e.g. dmp*x*=*N*) to the next level of the memory hierarchy. The value in *N* is representative of processor cycles and is approximate due to its dependency on a clean or dirty line, read or write miss, etc. Note specify *N* as a positive integer. Setting *N*=0 specifies that **no** cache exist at that level.

`-LNO:cs1=`*N*`. cs2=`*N*`, cs3=`*N*`, cs4=`*N*
: The compiler is instructed to set the cache sizes. The value in *N* must include a suffix letter of *k* or *K* to indicate Kbytes or *m* or *M* to indicate Mbytes. Note specify *N* as a positive integer. Setting *N*=0 specifies that **no** cache exist at that level.

  Note:

  - primary cache is represented by **cs1**
  - secondary cache is represented by **cs2**
  - memory is represented by **cs3**
  - disk is represented by **cs4**

  Invoking the compiler with '`-LIST:all_options=ON`' will emit the default cache sizes of the host system being used for compilation. Cache sizes for each level of cache and memory are system dependent. The default is *N*=0 for each level of cache.

`-LNO:is_mem1=`*ON*`|`*OFF*`, is_mem2=`*ON*`|`*OFF*`, is_mem3=`*ON*`|`*OFF*`, is_mem4=`*ON*`|`*OFF*
: The compiler is instructed to setup the memory model hierarchy as cache or memory.

An attempt to perform loop blocking is permitted once the memory model has been structured. Note specify loop blocking suitable for memory and **not** for cache. No prefetching is performed and all prefetching options are disregarded.

Note LNO suboption '`-LNO:is_memx=ON|OFF`' supersedes the corresponding suboptions '`-LNO:assocx=N`' and '`-LNO:cmpx=N,...,dmpx=N`'.

`-LNO:ls1=N, ls2=N, ls3=N, ls4=N`

The compiler is instructed to set the cache line sizes. The value in N is the number of bytes in a cache line. The size of the cache line dictates the number of bytes moved from one memory hierarchy on a miss at this level to another memory hierarchy level. Note specify N as a positive integer. Setting N=0 specifies that **no** cache exist at that level.

The following **LNO Translation Lookaside Buffer Suboptions** allows the user to specify the translation lookaside buffer (TLB) characteristics. The suboption arguments are specified under the assumption that the cache for the page table is fully associative.

`-LNO:ps1=N, ps2=N, ps3=N, ps4=N`

The compiler is instructed to set the page table size. The value in N is the number of bytes in the page. Note specify N as a positive integer. The default for N is dependent on the target system hardware.

`-LNO:tlb1=N, tlb2=N, tlb3=N, tlb4=N`

The compiler is instructed to set the number of entries, N, in the TLB for a selected memory hierarchy level. Note specify N as a positive integer. The default for N is dependent on the target system hardware.

`-LNO:tlbcmp1=N, tlbcmp2=N, tlbcmp3=N, tlbcmp4=N, tlbdmp1=N, tlbdmp2=N, tlbdmp3=N, tlbdmp4=N`

The compiler is instructed to set the time to service a clean TLB miss (e.g. tlbcmp$x$=N) or a dirty TLB miss (e.g. dmp$x$=N) to the next level of the memory hierarchy. The value in N is representative of processor cycles and is approximate. Note specify N as a positive integer. The default for N is dependent on the target system hardware.

The following **LNO Prefetch Suboptions** allows the user to specify prefetch operations.

`-LNO:pf1=ON|OFF, pf2=ON|OFF, pf3=ON|OFF, pf4=ON|OFF`

The compiler is instructed to turn ON/OFF prefetching for specified cache levels.

`-LNO:prefetch=0|1|2|3`

The compiler is instructed to set the prefetching level. The flag can be set to:

0 Instructs the compiler to suppress prefetching.

1 Instructs the compiler to set prefetching only for arrays that are referenced in every loop iteration.

2 Instructs the compiler to turn prefetching *ON* disregarding the above restriction.

3 Instructs the compiler to perform aggressive prefetching.

The defaulti is '`-LNO:prefetch=2`'

`-LNO:prefetch_ahead=`*N*
The compiler is instructed to prefetch a set number, *N*, of cache lines ahead of the reference. Note specify *N* as a positive number. The default is '`-LNO:prefetch_ahead=2`'.

`-LNO:prefetch_manual=`*ON*|*OFF*
The compiler is instructed to allow directives specifying manual prefetches. If '`-LNO:prefetch_manual=OFF`' the directives are ignored. The default is '`-LNO:prefetch_manual=ON`'.

3.10 Options Controlling the Preprocessor

These options control the C preprocessor, which is run on each C source file before actual compilation.

If you specify the '`-E`' option nothing is done except preprocessing. See [option '`-E`'], page 33, in Option Controlling the Kind of Output section. Some of these options make sense only together with '`-E`' because they cause the preprocessor output to be unsuitable for actual compilation.

`-A` *predicate=answer*
`-A -`*predicate=answer*
Make an assertion with the predicate *predicate* and answer *answer*. Option '`-A -`*predicate=answer*' cancels an assertion with the predicate *predicate* and answer *answer*.

`-C` (C Only)
Do not discard comments after preprocessing. All comments are passed through to the output file, except for comments in processed directives, which are deleted along with the directive.

The user will note the following side effects when using '`-C`'; it causes the preprocessor to treat comments as tokens in their own right. For example, comments appearing at the start of what would be a directive line have the effect of turning that line into an ordinary source line, since the first token on the line is no longer a '`#`'.

`-cpp` Instructs the compiler to pass all input source code through the GCC preprocessor (i.e. the C preprocessor, cpp) before compiling, regardless of the '`filename`' suffix.

Note options '`-ftpp`', '`-E`', and '`-nocpp`' provide additional control of preprocessing. The '`-macro-expand`' option can be specified to enable macro expansion.

The default is the pass input files with suffix '`.F`' or '`.F90`' through the C preprocessor (i.e. cpp). Preprocessing is **not** performed on files that have suffixes '`.f`' or '`.f90`'.

`-d`*CHAR* The compiler is instructed to generate and write a specified list to the standard output file. Note *CHAR* is a sequence of one or more of the following characters, and must not be preceded by a space. Other characters are interpreted by the compiler proper, or reserved for future versions of Open64, and so are silently ignored. If you specify characters whose behavior conflicts, the result is undefined.

-dD Generate a list of non-predefined macro directives.

-dI Output `#include` directives in addition to preprocessor results.

-dM Generate a list of directives for all macros.

-dN Generate a list of all macro names defined.

`-D`*name* Predefine *name* as a macro, with definition `1`. The *name* must be declared as a logical constant in the source files and will be set to true.

`-D`*name*`=`*definition*

`-D`*var*`=`*definition*`[,`*var*`=`*definition*`,...]`

The contents of *definition* are tokenized and processed as if they appeared during translation phase three in a '`#define`' directive. In particular, the definition will be truncated by embedded newline characters. The preprocessor assigns values to the constants preempting values assigned within the source files. When the option contains an "=" sign the value on the right must be an integer and the *name* on the left must be declared as an integer constant in the source files.

Note if the definition (i.e. *def*) is **not** specified a *1* is used. See ['`-U`*name*'], page 82, for information on undefined variables. '`-D`' and '`-U`' options are processed in the order they are given on the command line.

If you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax.

If you wish to define a function-like macro on the command line, write its argument list with surrounding parentheses before the equals sign (if any). Parentheses are meaningful to most shells, so you will need to quote the option. For example, with `sh` and `csh`, '`-D'`*name*`(`*args...*`)=`*definition*`'`'.

`-fe` Instruct the compiler to halt after the prepreocessor is run (i.e. the compiler front-end).

`-fpreprocessed` (C/C++ Only)

`-fno-preprocessed` (C/C++ Only)

Instructs the compiler that the input source file has been preprocessed. '`-fno-preprocessed`' specifies that the input source file has not been preprocessed.

`-ftpp` (Fortran Only)
: Fortran source files are processed by the Fortran source preprocessor before compiling.

  By default, only files with suffix `.F`, `.F90`, or `.F95` are processed by the C source preprocessor, '`cpp`'. Source files with suffix `.f`, `.f90`, or `.f95` are **not** processed by the preprocessor

`-M`
: Instead of outputting the result of preprocessing, output a rule suitable for `make` describing the dependencies of the main source file. The preprocessor outputs one `make` rule containing the object file name for that source file, a colon, and the names of all the included files, including those coming from '`-include`' or '`-imacros`' command line options.

  Unless specified explicitly (with '`-MT`' or '`-MQ`'), the object file name consists of the basename of the source file with any suffix replaced with object file suffix. If there are many included files then the rule is split into several lines using '\'-newline. The rule has no commands.

  To avoid mixing such debug output with the dependency rules you should explicitly specify the dependency output file with '`-MF`', or use an environment variable like `DEPENDENCIES_OUTPUT`. Debug output will still be sent to the regular output stream as normal.

  Passing '`-M`' to the driver implies '`-E`', and suppresses warnings with an implicit '`-w`'.

`-macro-expand` (Fortran Only)
: The preprocessor performs macro expansion throughout each Fortran source file. When option is **not** specified macro expansion is limited to preprocessor `#` directives only.

`-MD`
: '`-MD`' is equivalent to '`-M -MF` *`file`*', except that '`-E`' is not implied. The driver determines *file* based on whether an '`-o`' option is given. If it is, the driver uses its argument but with a suffix of '`.d`', otherwise it takes the basename of the input file and applies a '`.d`' suffix.

  If '`-MD`' is used in conjunction with '`-E`', any '`-o`' switch is understood to specify the dependency output file (see [-MF], page 80), but if used without '`-E`', each '`-o`' is understood to specify a target object file.

  Since '`-E`' is not implied, '`-MD`' can be used to generate a dependency output file as a side-effect of the compilation process.

`-MDtarget` *`filename`* (Fortran Only)
: Use *filename* as the target for makefile dependencies. Used in conjunction with the option '`-MDupdate`'.

`-MDupdate` *`filename`* (Fortran Only)
: Updates makefile dependencies in *filename*.

`-MF` *`filename`*
: When used with '`-M`' or '`-MM`', specifies a *filename* to write the dependencies to. If no '`-MF`' switch is given the preprocessor sends the rules to the same place it would have sent preprocessed output.

When used with the driver options '`-MD`' or '`-MMD`', '`-MF`' overrides the default dependency output file.

`-MG` In conjunction with an option such as '`-M`' or '`-MM`' requesting dependency generation, '`-MG`' assumes missing header files are generated files and adds them to the dependency list without raising an error. The dependency filename is taken directly from the `#include` directive without prepending any path. '`-MG`' also suppresses preprocessed output, as a missing header file renders this useless.

This feature is used in automatic updating of makefiles.

`-MM` Like '`-M`' but do not mention header files that are found in system header directories, nor header files that are included, directly or indirectly, from such a header.

This implies that the choice of angle brackets or double quotes in an '`#include`' directive does not in itself determine whether that header will appear in '`-MM`' dependency output. This is a slight change in semantics from GCC versions 3.0 and earlier.

`-MMD` Like '`-MD`' except mention only user header files, not system header files.

`-MP` This option instructs CPP to add a phony target for each dependency other than the main file, causing each to depend on nothing. These dummy rules work around errors `make` gives if you remove header files without updating the '`Makefile`' to match.

This is typical output:

```
test.o: test.c test.h

test.h:
```

`-MQ` *target* (C/C++ Only)
Same as '`-MT`', but it quotes any characters which are special to Make. '`-MQ '$(objpfx)foo.o'`' gives

```
$$(objpfx)foo.o: foo.c
```

The default target is automatically quoted, as if it were given with '`-MQ`'.

`-MT` *target* (C/C++ Only)
Change the target of the rule emitted by dependency generation. By default CPP takes the name of the main input file, including any path, deletes any file suffix such as '`.c`', and appends the platform's usual object suffix. The result is the target.

An '`-MT`' option will set the target to be exactly the string you specify. If you want multiple targets, you can specify them as a single argument to '`-MT`', or use multiple '`-MT`' options.

For example, '`-MT '$(objpfx)foo.o'`' might give

```
$(objpfx)foo.o: foo.c
```

`-nocpp` (Fortran Only)
Do **not** run the source preprocessor (cpp) on all input source files. See '`-cpp`', '`-E`', and '`ftpp`' for more information on controlling preprocessing.

`-no-gcc` (Fortran Only)
: Disables predefined preprocessor macros, e.g. `__GNUC__`

`-P`
: Inhibit generation of linemarkers in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code, and will be sent to a program which might be confused by the linemarkers. Used with option '`-E`', See [option '`-E`' in Option Controlling the Kind of Output section], page 33.

`-U`*name*
: Cancel any previous definition of *name*, either built in or provided with a '`-D`' option.

`-Wp,`*option*`,...`
: You can use '`-Wp,`*option*' to bypass the compiler driver and pass *option* directly through to the preprocessor. If *option* contains commas, it is split into multiple options at the commas. However, many options are modified, translated or interpreted by the compiler driver before being passed to the preprocessor, and '`-Wp`' forcibly bypasses this phase. The preprocessor's direct interface is undocumented and subject to change, so whenever possible you should avoid using '`-Wp`' and let the driver handle the options instead.

3.11 Passing Options to the Assembler

You can pass options to the assembler.

`-fno-asm` (C/C++ Only)
: Do not recognize `asm`, `inline` or `typeof` as a keyword, so that code can use these words as identifiers. You can use the keywords `__asm__`, `__inline__` and `__typeof__` instead. '`-ansi`' implies '`-fno-asm`'.

  In C++, this switch only affects the `typeof` keyword, since `asm` and `inline` are standard keywords. You may want to use the '`-fno-gnu-keywords`' flag instead, which has the same effect. In C99 mode ('`-std=c99`' or '`-std=gnu99`'), this switch only affects the `asm` and `typeof` keywords, since `inline` is a standard keyword in ISO C99.

`-Wa,`*option*`,...`
: Pass *option* as an option to the assembler. If *option* contains commas, it is split into multiple options at the commas.

3.12 Options Controlling the Linker and Libraries

These options come into play when the compiler links object files into an executable output file. They are meaningless if the compiler is not doing a link step.

object-file-name
: A file name that does not end in a special recognized suffix is considered to name an object file or library. (Object files are distinguished from libraries by the linker according to the file contents.) If linking is done, these object files are used as input to the linker.

`-ar`
: Instructs the compiler to create an archive file instead of a shared object or executable file. To specify an archive file name use '`-o` *filename*'. Before creating

the archive file, template entities needed by the archived objects are instantiated. When `openCC` invokes the compiler with '`-ar`' specified, it implicitly passes options '`-c`' and '`-r`' to the compiler. In addition the *filename* of the archive and object files being created are passed. Option '`-WR,`*`option-list`*'is used to pass required options that can be used concurrently with the '`-c`' option.

Note options specified with the '`-WR,`*`option-list`*' must include all objects that will be incorporated in the archive, otherwise prelinked internal errors will be emitted. In the following example:

```
openCC -ar -WR,-v -o liba.a file1.o file2.o file3.o
```

'`liba.a`' is an archive which incorporates files '`file1.o`', '`file2.o`', and '`file3.o`'. Object files '`file1.o`', '`file2.o`', and '`file3.o`' are prelinked to instantiate all required template entities. Then the `ar -r -c -v liba.a file1.o file2.o file3.o` command is invoked. Even if only '`file3.o`' needs to be replaced in '`liba.a`', all three object files must be specified.

`-c`
`-S`
`-E` If any of these options is used, then the linker is not run, and object file names should not be used as arguments. See Section 3.2 [Overall Options], page 32.

`-ffast-stdlib`
`-fno-fast-stdlib`
Instructs the compiler to generate code that links against special versions of some standard library routines (e.g., fast versions of the standard library routines). Specifying '`-fno-fast-stdlib`' instructs the compiler **not** to generate code that links against fast versions of standard library routines. Linking code with '`-fno-fast-stdlib`' that has **not** been compiled using this flag may emit linker errors. Note specifying '`-ffsat-stdlib`' implies '`-OPT:fast_stdlib=ON`'. The default is '`-ffast-stdlib`'.

`-H` Print the name of each header file used, in addition to other normal activities. Each name is indented to show how deep in the '`#include`' stack it is. Precompiled header files are also printed, even if they are found to be invalid; an invalid precompiled header file is printed with '`...x`' and a valid one with '`...!`' .

`-l` *`library`*
Search the library named *library* when linking. (The second alternative with the library as a separate argument is only for POSIX compliance and is not recommended.)

It makes a difference where in the command you write this option; the linker searches and processes libraries and object files in the order they are specified. Thus, '`foo.o -lz bar.o`' searches library '`z`' after file '`foo.o`' but before '`bar.o`'. If '`bar.o`' refers to functions in '`z`', those functions may not be loaded.

The linker searches a standard list of directories for the library, which is actually a file named '`lib`*`library`*`.a`'. The linker then uses this file as if it had been specified precisely by name.

The directories searched include several standard system directories plus any that you specify with '`-L`'.

Normally the files found this way are library files—archive files whose members are object files. The linker handles an archive file by scanning through it for members which define symbols that have so far been referenced but not defined. But if the file that is found is an ordinary object file, it is linked in the usual fashion. The only difference between using an '`-l`' option and specifying a file name is that '`-l`' surrounds *library* with '`lib`' and '`.a`' and searches several directories.

`-nostartfiles`
Do not use the standard system startup files when linking. The standard system libraries are used normally, unless '`-nostdlib`' or '`-nodefaultlibs`' is used.

`-nodefaultlibs`
Do not use the standard system libraries when linking. Only the libraries you specify will be passed to the linker. The standard startup files are used normally, unless '`-nostartfiles`' is used. The compiler may generate calls to `memcmp`, `memset`, `memcpy` and `memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

`-nostdinc`
Instructs the compiler to skip the standard directory (i.e. '`/usr/include/`') when searching for `#include` files and files named in the `INCLUDE` statements.

`-nostdinc++` (C++ Only)
Do not search for header files in the standard directories specific to C++, but do still search the other standard directories. (This option is used when building the C++ library.)

`-nostdlib`
Do not use the standard system startup files or libraries when linking. No startup files and only the libraries you specify will be passed to the linker. The compiler may generate calls to `memcmp`, `memset`, `memcpy` and `memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

One of the standard libraries bypassed by '`-nostdlib`' and '`-nodefaultlibs`' is '`libgcc.a`', a library of internal subroutines that the compiler uses to overcome shortcomings of particular machines, or special needs for some languages. In most cases, you need '`libgcc.a`' even when you want to avoid other standard libraries. In other words, when you specify '`-nostdlib`' or '`-nodefaultlibs`' you should usually specify '`-lgcc`' as well. This ensures that you have no unresolved references to internal GCC library subroutines.

`-objectlist` *`filename`*
Instructs the compiler to open *filename* to retrieve the list of files to be linked.

`-shared` Produce a shared object which can then be linked with other objects to form an executable. Not all systems support this option. For predictable results,

you must also specify the same set of options that were used to generate code ('`-fpic`', '`-fPIC`', or model suboptions) when you specify this option.[1]

Note care should be taken mixing '`-shared`' with '`-IPA`' options. Interprocedural analysis assumes the compiler sees the entire program to perform optimizations. Incorrect programs can be produced when only part of a program is placed in a shared library.

`-shared-libgcc`
`-static-libgcc`

On systems that provide '`libgcc`' as a shared library, these options force the use of either the shared or static version respectively. If no shared version of '`libgcc`' was built when the compiler was configured, these options have no effect.

There are several situations in which an application should use the shared '`libgcc`' instead of the static version. The most common of these is when the application wishes to throw and catch exceptions across different shared libraries. In that case, each of the libraries as well as the application itself should use the shared '`libgcc`'.

Therefore, the `openCC` drivers automatically add '`-shared-libgcc`' whenever you build a shared library or a main executable, because C++ programs typically use exceptions, so this is the right thing to do.

If, instead, you use the `opencc` driver to create shared libraries, you may find that they will not always be linked with the shared '`libgcc`'. If the compiler finds, at its configuration time, that you have a non-GNU linker or a GNU linker that does not support option '`--eh-frame-hdr`', it will link the shared version of '`libgcc`' into shared libraries by default. Otherwise, it will take advantage of the linker and optimize away the linking with the shared version of '`libgcc`', linking with the static version of libgcc by default. This allows exceptions to propagate through such shared libraries, without incurring relocation costs at library load time.

However, if a library or main executable is supposed to throw or catch exceptions, you must link it using the `openCC`driver or using the option '`-shared-libgcc`', such that it is linked with the shared '`libgcc`'.

`-static`
`--static` On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect. '`--static`' is equivalent to '`-static`', except '`--static`' does **not** incite the compiler to emit warnings regarding possible confusion with '`-static-data`'.

`-static-data` (Fortran Only)

Instructs the compiler to statically allocate all local variables which are initially set to zero and will exist for the life of the program. Global data is allocated as

[1] On some systems, '`gcc -shared`' needs to build supplementary stub code for constructors to work. On multi-libbed systems, '`gcc -shared`' must select the correct support libraries to link against. Failing to supply the correct flags may lead to subtle defects. Supplying them in cases where they are not necessary is innocuous.

part of the compiled object file. The total size of any object file is limited to 2 GB. The total size of a program loaded using multiple object files are allowed to exceed the 2 GB limit. Multiple common blocks each within the 2 GB size limit can be declared.

The '`-static-data`' cannot be specified when compiling an external routine that is called by a program which contains parallel loops targeting a multiprocessor system. Static and multiprocessor compiled object files can be mixed in the same executable, but a static routine cannot be called from within a parallel region.

Note option '`-static-data`' is useful when porting programs from legacy systems in which all variables are allocated as static.

`-stdinc` Instructs the compiler to use a predifined include search path list.

`-symbolic`

Bind references to global symbols when building a shared object. Warn about any unresolved references (unless overridden by the link editor option '`-Xlinker -z -Xlinker defs`'). Only a few systems support this option.

`-Xlinker` *option*

Pass *option* as an option to the linker. You can use this to supply system-specific linker options which the compiler does not know how to recognize.

If you want to pass an option that takes an argument, you must use '`-Xlinker`' twice, once for the option and once for the argument. For example, to pass '`-assert definitions`', you must write '`-Xlinker -assert -Xlinker definitions`'. It does not work to write '`-Xlinker "-assert definitions"`', because this passes the entire string as a single argument, which is not what the linker expects.

`-Wl,`*option*`,...`

Pass *option* as an option to the linker. If *option* contains commas, it is split into multiple options at the commas.

3.13 Options for Code Generation Conventions

These machine-independent options control the interface conventions used in code generation.

`-CG:`*question*`=`*answer*

This group of code generation option controls the optimizations and transformations of the instruction level code generator.

`-CG:cflow=`*ON*`|`*OFF*

'`-CG:cflow=`*OFF*' disables control flow optimization in the code generation. The default is '`-CG:cflow=`*ON*'.

`-CG:cmp_peep=`*ON*`|`*OFF*

Instructs the compiler to perform aggressive load execution peeps on compare operations. The default is '`CG:cmp_peep=OFF`'.

`-CG:compute_to=`*on*`|`*off*

Instructs the compiler to allow local code motion to take advantage of pipeline optimizations. For example: specify in conjunction with option '`-march=barcelona`', i.e.when targeting versions of the Quad-Core AMD® Opteron™ and greater. The default is '`CG:compute_to=OFF`'

`-CG:cse_regs=`*N*

When performing common subexpression elimination, instructs the compiler code generator that there are *N* extra integer registers available (i.e. in excess of the number provided by the CPU). *N* can be positive, negative, or zero. The default is positive infinity. See ['`-CG:sse_cse_regs=`*N*'], page 89.

`-CG:gcm=`*ON*`|`*OFF*

'`-CG:gcm=`*OFF*' instructs the compiler to disable the instruction level global code motion optimization phase. The default is '`-CG:gcm=`*ON*'.

`-CG:inflate_reg_request=`*N*

Instructs the the local register allocator to increase its register request by *N* percent for innermost loops. The default is '`-CG:inflate_reg_request=`*0*'.

`-CG:load_exe=`*N*

The parameter *N* must be a non-negative integer which specifies the threshold for the compiler to consider folding a memory load operation directly into its subsequent use in an arithmetic instruction (thereby eliminating the memory load operation). If *N=0* this folding optimization is not performed (in other words, the optimization is turned off). If the number of times the result of the memory load is used exceeds the value of *N*, then the folding optimization is not performed. For example, if *N=1* this optimization is performed only when the result of the memory load has only one use. The default value of *N* varies with target processor and source language.

`-CG:local_sched_alg=`*0*`|`*1*`|`*2*

This option selects the basic block instruction scheduling algorithm.

- To perform backward scheduling (i.e. where instructions are scheduled from the bottom to the top of the basic block) select *0*.
- To perform forward scheduling select *1*.
- To schedule the instruction twice (i.e. once in the forward direction and once in the backward direction) and take the optimal of the two schedules.

The default value for this option is determined by the x86 Open64 compiler during compilation.

`-CG:locs_best=ON|OFF`
: When enabled the local instruction scheduler is run several times using different heuristics. The optimal schedule generated is selected. Note this option supersedes options which control local instruction scheduling, e.g. '`-CG:local_sched_alg`' and '`-CG:locs_shallow_depth`'. The default is '`-CG:locs_best=OFF`'.

`-CG:locs_reduce_prefetch=ON|OFF`
: Setting to *ON* instructs the compiler to delete prefetch instructions that cannot be scheduled into unused processor cycles. Note this occurs only for backward instruction scheduling. The default is '`-CG:locs_reduce_prefetch=OFF`'.

`-CG:locs_shallow_depth=ON|OFF`
: *ON* instructs the compiler to give priority to instructions that have shallow depths in the dependence graph when performing local instruction scheduling to reduce register usage. The default is '`-CG:locs_shallow_depth=OFF`'.

`-CG:movnti=N`
: When writing memory blocks of size larger than N KB ordinary stores will be transformed to non-temporal stores. The default is N=1000.

`-CG:p2align=ON|OFF`
: When *ON* instructs the compiler to align loop heads to 64-byte boundaries. The default is '`-CG:p2align=OFF`'.

`-CG:p2align_freq=N`
: Values for N specify the execution frequency threshold. The compiler will perform branch target alignments when the execution frequency is equal to or greater than the specified threshold, N. Note this option is only valid when using feedback-direct compilation. The default is N=1000.

`-CG:post_local_sched=ON|OFF`
: When *ON*, enables the loacal scheduler phase after register allocation. The default is '`-CG:post_local_sched=ON`'.

`-CG:pre_local_sched=ON|OFF`
: When *ON*, enables the local scheduler phase before register allocation. The default is '`-CG:pre_local_sched=ON`'.

`-CG:prefer_legacy_regs=ON|OFF`
: When enabled the compiler register allocator is instructed to use the first 8 integer and SSE registers when possible. Starting with the last assigned register and work upward in most recently assigned fashion where available. Note instructions which use these registers have smaller instruction sizes. For example, the lower 8 registers on x86-64 do **not** use the `rex` byte. The default is '`-CG:prefer_legacy_regs=OFF`'.

`-CG:prefetch=`*ON*`|`*OFF*
: When '`-CG:prefetch=ON`' prefetch instructions are generated in the code generator. Note both '`-CG:prefetch=`*OFF*' and '`-LNO:prefetch=`*0*' disable the generation of prefetch instructions. Only '`-LNO:prefetch=`*0*' affects loop-nesting optimizations that rely on prefetch. The default is '`-CG:prefetch=`*OFF*'.

`-CG:ptr_load_use=`*N*
: The code generator increases the latency between an instruction that loads a pointer and an instruction that uses the pointer by *N* cycles. Ordinarily, it is advantageous to load pointers as fast as possible so the dependent memory instructions can begin execution. However, the additional latency will force the instruction scheduler to schedule the *load pointer* earlier. Note the *load pointer* instructions include load-execute instructions which compute pointer results. The default is '`-CG:ptr_load_use=`*4*'.

`-CG:push_pop_int_saved_regs=`*ON*`|`*OFF*
: When *ON*, the compiler generates push and pop instructions to save integer callee-saved registers at function prologues and epilogues. The push and pop instructions replace mov instructions to and from memory locations based off the stack pointer. The default is '`-CG:push_pop_int_saved_regs=`*OFF*'. When the specified target is the Quad-Core AMD Opteron^{TM} processor the default is '`-CG:push_pop_int_saved_regs=`*ON*'.

`-CG:sse_cse_regs=`*N*
: When performing common subexpression elimination, instructs the compiler code generator that there are *N* extra SSE registers available (i.e. in excess of the number provided by the CPU). *N* can be positive, negative, or zero. The default is positive infinity. See ['`-CG:cse_regs=`*N*'], page 87.

`-CG:unroll_fb_req=`*ON*`|`*OFF*
: The compiler is instructed to override cold code motion to keep the code generator from adding control flow in unrolled loops. The default is '`-CG:unroll_fb_req=OFF`'.

`-CG:use_prefetchnta=`*ON*`|`*OFF*
: When enabled prefetching is performed on non-temporal data at all levels of the cache hierarchy. Note for data streaming situations when the data will not need to be reused soon. The default is '`-CG:use_prefetchnta=`*OFF*'.

`-CG:use_test=`*ON*`|`*OFF*
: When enabled the code generator is forced to use `TEST` instruction instead of the `CMP` instruction. The default is '`-CG:use_test=`*OFF*'.

`-GRA:`*question=answer*
: Global register allocation (GRA) is the process of multiplexing a large number of target program variables onto a small number of CPU registers. The code

generator implements global register allocation when certain optimization levels are specified.

`-GRA:home=`*`ON`*`|`*`OFF`*

Instructs the compiler to perform a rematerialization optimization for non-local user variables in the register allocator. The default is '`-GRA:home=ON`'.

`-GRA:optimize_boundary=`*`ON`*`|`*`OFF`*

Instructs the compiler to permit the register allocator to assign the same register to different variables in the same basic-block. The default is '`-GRA:optimize_boundary=OFF`'.

`-GRA:prioritize_by_density=`*`ON`*`|`*`OFF`*

Instructs the compiler's GRA to prioritize register assignments to variables based on the variable's reference count density (number of times the variables are referenced in a local region of interest) and not on their global reference count. The default is '`-GRA:prioritize_by_density=OFF`'.

`-GRA:unspill=`*`ON`*`|`*`OFF`*

The compiler is instructed to mitigate existing and suboptimal boundary conditions between global register allocation and local register allocation by unspilling register candidates which were really available at those boundary conditions. The default is '`-GRA:unspill=OFF`'.

3.14 Specifying Target Environment and Machine

x86 Open64 provides options that will switch to another cross-compiler or target environment. The target environment is the system upon which the executable code will be run.

`-TENV:`*`question=answer`*

These options control the target environment assumed and/or produced by the compiler.

`-TENV:frame_pointer=`*`ON`*`|`*`OFF`*

Setting this option to *ON* tells the compiler to use the frame pointer register to address local variables in the function stack frame. Generally, if the compiler determines that the stack pointer is fixed it will use the stack pointer to address local variables throughout the function invocation in place of the frame pointer. This frees up the frame pointer for other purposes.

The default is *ON* for C/C++ and *OFF* for Fortran. This flag defaults to ON for C/C++ because the exception handling mechanism relies on the frame pointer register being used to address local variables. This flag can be turned OFF for C/C++ programs that do **not** generate exceptions.

`-TENV:simd_dmask=`*`ON`*`|`*`OFF`*

When set to *OFF* the SIMD floating-point denormalized-operation exception is unmasks. The default is *ON*

`-TENV:simd_imask=ON|OFF`
When set to *OFF* the SIMD floating-point invalid-operation exception is unmasks. The default is *ON*

`-TENV:simd_omask=ON|OFF`
When set to *OFF* the SIMD floating-point overflow exception is unmasks. The default is *ON*

`-TENV:simd_pmask=ON|OFF`
When set to *OFF* the SIMD floating-point precision exception is unmasks. The default is *ON*

`-TENV:simd_umask=ON|OFF`
When set to *OFF* the SIMD floating-point underflow exception is unmasks. The default is *ON*

`-TENV:simd_zmask=ON|OFF`
When set to *OFF* the SIMD floating-point zero-divide exception is unmasks. The default is *ON*

`-TENV:X=0,1,2,3,4`
Use this option to specify the level of enabled exceptions which will be granted for purposes of performing speculative code motion. The default level of enablement is *1* for all optimization levels. Instructions will **not** be speculated or moved above a branch by the optimizer unless all exceptions generated by the move are disabled.

0 Speculative code motion may **not** be performed.

1 Safe speculative code motion may be performed. IEEE-754 underflow exceptions must be disabled.

2 Disables all IEEE-754 exceptions except divide-by-zero

3 Disables all IEEE-754 exceptions including divide-by-zero.

4 Memory exceptions are disabled.

3.14.1 Hardware Models and Configurations

Earlier we discussed the option '`-TENV:`' which chooses among different environments for completely different target machines.

In addition, each of these target machine types can have its own special options, starting with '`-m`', to choose among various hardware models or configurations—for example, 80386 vs AMD Opteron^{TM}, floating coprocessor or none. A single installed version of the compiler can compile for any model or configuration, according to the options specified.

Some configurations of the compiler also support additional special options, usually for compatibility with other compilers on the same platform.

These '`-m`' options are defined for the i386 and x86-64 family of computers in 32-bit and 64-bit environments:

`-march=`*cpu-type*
`-mtune=`*cpu-type*
`-mcpu=`*cpu-type*

Generate instructions for the machine type *cpu-type*. The choices for *cpu-type* are the same as for '`-march`', '`-mtune`', and '`-mcpu`'. Moreover, specifying '`-march=`*cpu-type*' implies '`-mtune=`*cpu-type*'.

Tune to *cpu-type* everything applicable about the generated code, except for the ABI and the set of available instructions. The choices for *cpu-type* are:

anyx86 Produce code optimized for the most common x86-32/x86-64 processors. If you know the CPU on which your code will run, then you should use the corresponding '`-march`' option instead of '`-march=anyx86`'. But, if you do not know exactly what CPU users of your application will have, then you should use this option.

As new processors are deployed in the marketplace, the behavior of this option will change. Therefore, if you upgrade to a newer version of x86 Open64, the code generated option will change to reflect the processors that were most common when that version of Open64 was released.

auto This selects the CPU to tune for at compilation time by determining the processor type of the compiling machine. Using '`-march=auto`' will produce code optimized for the local machine under the constraints of the selected instruction set. Using '`-march=auto`' will enable all instruction subsets supported by the local machine (hence the result might not run on different machines).

athlon The original AMD Athlon^{TM} Processor.

athlon64 The AMD Athlon^{TM} 64 is the K8 core based CPUs, and eighth-generation processor featuring x86-64 technology.

athlon64fx The AMD Athlon^{TM} 64 FX is the K8 core based CPUs with dual cores.

barcelona The third-generation AMD Opteron^{TM} Processor. The Quad-Core AMD Opteron^{TM} K10H core based CPUs with x86-64 instruction set support.

core The Intel Core 2 processor with Intel64 support.

em64t Intel x86-64 instruction set support.

opteron The second-generation AMD Opteron^{TM} with x86-64 instruction set support.

pentium4 Intel Pentium 4 CPU with MMX, SSE and SSE2 instruction set support.

wolfdale Intel's dual-core Core 2 Duo processors.

xeon Intel's Xeon based CPU's with x86-64 instruction set support.

While picking a specific *cpu-type* will schedule things appropriately for that particular chip, the compiler will not generate any code that does not run on the i386 without the '`-march=cpu-type`' option being used.

`-msse`
`-mno-sse`
`-msse2`
`-mno-sse2`
`-msse3`
`-mno-sse3`
`-msse4a`
`-mno-sse4a`
`-m3dnow`
`-mno-3dnow`

These switches enable or disable the use of instructions in the MMX, SSE, SSE2, SSE3, SSE4a or 3DNow!^{TM} extended instruction sets. These extensions are also available as built-in functions.

These options will enable the compiler to use these extended instructions in generated code.

`-msse`
`-mno-sse` Enables/disables the use of SSE2 and SSE3 instructions. Note disabling SSE2 instructions when specifying '`-m64`' will emit a warning message.

`-msse2`
`-mno-sse2`
Enables/disables the use of SSE2 instructions. Note disabling SSE2 instructions when specifying '`-m64`' will emit a warning message. Specifying either '`-m64`' or '`-m32`' enables this option.

`-msse3`
`-mno-sse3`
Enables/disables the use of SSE3 instructions. Specifying options '`-march=barcelona`', '`-march=em64t`', or '`-march=core`' enables this option. The default is '`-mno-msse3`'.

`-msse4a`
`-mno-sse4a`
Enables/disables the use of SSE4a instructions. Specifying options '`-march=barcelona`' enables this option. The default is '`-mno-sse4a`'.

`-m3dnow`
`-mno-m3dnow`
Enables/disables the use of 3DNow!^{TM} instructions. The default is '`-mno-m3dnow`'.

Note applications which perform runtime CPU detection must compile separate files for each supported architecture, using the appropriate flags. In particular,

the file containing the CPU detection code should be compiled without these options.

`-m32`
`-m64` Generate code for a 32-bit or 64-bit environment. The 32-bit environment sets int, long and pointer to 32 bits and generates code that runs on any i386 system. The compiler generates x86 or IA32 32-bit ABI. The 64-bit environment sets int to 32 bits and long and pointer to 64 bits and generates code for AMD's x86-64 architecture. The compiler generates AMD64, INTEL64, x86-64 64-bit ABI. The default on a 32-bit host is 32-bit ABI. The default on a 64-bit host is 64-bit ABI if the target platform specified is 64-bit, otherwise the default is 32-bit.

`-mcmodel=`*`small|medium`*
Generate code for the *small* or *medium* memory models. The compiler generates these models:

- Specifying *small* implies the program and its symbols must be linked in the lower 2 GB of the address space. Pointers are 64 bits. Programs can be statically or dynamically linked. This is the default memory model.
- Specifying *medium* implies the program is linked in the lower 2 GB of the address space but symbols can be located anywhere in the address space. Programs can be statically or dynamically linked, but building of shared libraries are not supported with the *medium* model.

Note most programs will execute using 32-bit code and data pointers defined by the *small* memory model. If a program requires 64-bit data pointers then *medium* must be selected. Currently the compiler does **not** support the large memory model.

3.15 Options to Control Diagnostic

Traditionally, diagnostic messages have been formatted irrespective of the output device's aspect (e.g. its width, ...). The options described below can be used to control the diagnostic messages formatting algorithm, e.g. how many characters per line, how often source location information should be reported, etc. Right now, only the C++ front end can honor these options. However it is expected, in the near future, that the remaining front ends would be able to digest them correctly.

`-C` (Fortran only)
Performs run-time array subscript range checking. A subscript range violation will trigger a fatal run-time error. Note if the environment variable `F90_BOUNDS_CHECK_ABORT` is set to *YES* the program aborts.

`-clist` (C Only)
`-CLIST:=`*`answer`* (C Only)
`-CLIST:`*`question`*`=`*`answer`* (C Only)
This option group is a C language diagnostic tool. Option group '`CLIST:`' instructs the compiler to emit internal program representation back into C code, after IPA inlining and loopnest transformations. The generated C code may

not always be compatible and is written to two files, a header file containing filescope declarations, and a file containing function definitions.

Option '`-clist`' is equivalent to '`-CLIST:=ON`'. Specifying any variation of option '`-CLIST:question=answer`' implies '`-clist`', (i.e. with the exception of '`CLIST:=OFF`'). The individual controls in this group are as follows:

`-clist` Equivalent to enabling '`-CLIST:`', i.e. '`-CLIST:=ON`'.

`-CLIST:=ON|OFF`
Option '`-CLIST:=ON`' instructs the compiler to emit internal program representation back into the C code. This option is implied if any of the '`-CLIST:=question=answer`' options are specified. '`-CLIST:=ON`' is equivalent to '`-clist`'.

`-CLIST:dotc_file=filename`
Instructs the compiler to write the program units to the specified source file *filename*. The default suffix for *filename* is *.w2c.c*.

`-CLIST:doth_file=filename`
Instructs the compiler to write the file-scope declarations to the specified file *filename*. The default suffix for *filename* is *.w2c.h*.

`-CLIST:emit_pfetch=ON|OFF`
Instructs the compiler to insert comments includes prefetch information in the transformed source file. The default is '`-CLIST:emit_pfetch=OFF`'.

`-CLIST:linelength=N`
Instructs the compiler to set the maximum line length to *N* characters. The default is a unlimited number of characters-per-line.

`-CLIST:show=ON|OFF`
Instructs the compiler to print the input and output file names to `stderr`. The default is '`-CLIST:show=ON`'.

`-ffortran-bounds-check` (Fortran Only)
Checks bounds. '`-ffortran-bounds-check`' is equivalent to option '`-C`'.

`-flist` (Fortran Only)
`-FLIST:=answer` (Fortran Only)
`-FLIST:question=answer` (Fortran Only)
This option group is a Fortran language diagnostic tool. Option group '`FLIST:`' instructs the compiler to emit internal program representation back into the Fortran code, after IPA inlining and loopnest transformations. The generated Fortran code may **not** always be compatible and compile successfully.

Option '`-flist`' is equivalent to '`-FLIST:=ON`'. Specifying any variation of option '`-FLIST:question=answer`' implies '`-flist`', (i.e. with the exception of '`FLIST:=OFF`'). The individual controls in this group are as follows:

`-flist` Equivalent to enabling '`-FLIST:`', i.e. '`-FLIST:=ON`'.

`-FLIST:=ON|OFF`
Option '`-FLIST:=ON`' instructs the compiler to emit internal program representation back into the Fortran code. This option is

implied if any of the '`-FLIST:=question=answer`' options are specified. '`-FLIST:=ON`' is equivalent to '`-flist`'.

`-FLIST:ansi_format=ON|OFF`
: Instructs the compiler to set ANSI format. When set to *ON*, the compiler is instructed to use a space instead of a tab for indentations and to write a maximum of 72 characters-per-line. The default is '`-FLIST:ansi_format=OFF`'.

`-FLIST:emit_pfetch=ON|OFF`
: Instructs the compiler to write prefetch information as comments in the transformed source file. The compiler will write **PREFETCH** in the listing to identify the reference being prefetched and will include:

  – the variable reference with an offset in bytes
  – an indication of read/write
  – a stride for each dimension
  – a number in the range for 1 to 3 (with 1 being the lowest) which reflects the confidence in the prefetch analysis

  The written comments appear after a read/write to a variable and note the identifier of the prefetch-spec for each level of the cache. The default is '`-FLIST:emit_pfetch=OFF`'.

`-FLIST:ftn_file=filename`
: Instructs the compiler to write the program to the file, *filename*. The default suffix for *filename* is *.w2f.f*.

`-FLIST:linelength=N`
: Instructs the compiler to set the maximum line length to *N* characters. The default is '`-FLIST:linelength=72`'.

`-FLIST:show=ON|OFF`
: Instructs the compiler to print the input and output file names to '`stderr`'. The default is *ON*.

`-fpermissive`
`-fno-permissive`
: Option '`-fpermissive`' will downgrade some diagnostics about nonconformant code from errors to warnings. Thus, using '`-fpermissive`' will allow some nonconforming code to compile. '`-fno-permissive`' maintains diagnostics about nonconformant code as errors.

  Using the '`-fpermissive`' flag will also let the compiler accept the code, by marking all function calls for which no declaration is visible at the time of definition of the template for later lookup at instantiation time, as if it were a dependent call. We do not recommend using -fpermissive to work around invalid code, and it will also only catch cases where functions in base classes are called, not where variables in base classes are used.

`-fullwarn`
: Instructs the compiler to generate comment level diagnostics. It may be beneficial to specify this option during program development. The default is to disable this diagnostic messaging.

`-pedantic-errors` (C Only)
: Issue all the errors demanded by strict ISO C and ISO C++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. For ISO C, follows the version of the ISO C standard specified by any '`-std`' option used.

  Valid ISO C and ISO C++ programs should compile properly with or without this option (though a rare few will require '`-ansi`' or a '`-std`' option specifying the required version of ISO C). However, without this option, certain GNU extensions and traditional C and C++ features are supported as well. With this option, they are rejected.

  '`-pedantic-errors`' does not cause error messages for use of the alternate keywords whose names begin and end with '`__`'. Pedantic errors are also disabled in the expression that follows `__extension__`. However, only system header files should use these escape routes; application programs should avoid them.

  Some users try to use '`-pedantic-errors`' to check programs for strict ISO C conformance. They soon find that it does not do quite what they want: it finds some non-ISO practices, but not all—only those for which ISO C *requires* a diagnostic, and some others for which diagnostics have been added.

  Where the standard specified with '`-std`' represents a GNU extended dialect of C, such as '`gnu89`' or '`gnu99`', there is a corresponding *base standard*, the version of ISO C on which the GNU extended dialect is based. Errors from '`-pedantic-errors`' are given where they are required by the base standard.

`-subverbose`
: This option is not supported by the compiler and is ignored if specified.

`-trapuv` Instructs the compiler to initialize variables to the value *NaN*, forcing a program crash if it uses uninitialized variables (i.e. traps uninitialized variables). Note '`-trapuv`' affects local scalar and array variables plus memory returns by `alloca()`. It does **not** influence the behavior of globals, `malloc()`ed memory or Fortran common variables.

`-zerouv` Instructs the compiler to initialize variables to zero. Note '`-zerouv`' affects local scalar and array variables plus memory returns by `alloca()`. It does **not** influence the behavior of globals, `malloc()`ed memory or Fortran common variables.

3.16 Options for Debugging Your Program

x86 Open64 has various special options that are used for evaluating and debugging your program:

`-dletters`
: This option specifies debugging dumps, into a file named '`dumpname`', during compilation at phases specified by *letters*. This is used for debugging the RTL-

based passes of the compiler. *dumpname* is generated from the name of the output file, if explicitly specified and it is not an executable, otherwise it is the basename of the source file.

`-dD` Generate a list of all non-predifined macro directives, at the end of preprocessing, in addition to normal output.

`-dI` Output `#include` directives in addition to preprocessor results.

`-dM` Generate a list of directives for all macros.

`-dN` Generate a list of all macro names defined.

`-fprofile-arcs`
Add code so that program flow *arcs* are instrumented. During execution the program records how many times each branch and call is executed and how many times it is taken or returns. When the compiled program exits it saves this data to a file called '*auxname*`.gcda`' for each source file. The data may be used for profile-directed optimizations ('`-fbranch-probabilities`'), or for test coverage analysis ('`-ftest-coverage`'). Each object file's *auxname* is generated from the name of the output file, if explicitly specified and it is not the final executable, otherwise it is the basename of the source file. In both cases any suffix is removed (e.g., '`foo.gcda`' for input file '`foo.c`', or '`dir/foo.gcda`' for output file specified as '`-o dir/foo.o`').

`-frandom-seed=`*string* (C/C++ Only)
This option provides a seed that the compiler uses when it would otherwise use random numbers. It is used to generate certain symbol names that have to be different in every compiled file. It is also used to place unique stamps in coverage data files and the object files that produce them. You can use the '`-frandom-seed`' option to produce reproducibly identical object files.

The *string* can be any character string and should be different for every file you compile.

`-ftest-coverage`
Produce a notes file that the `gcov` code-coverage utility can use to show program coverage. Each source file's note file is called '*auxname*`.gcno`'. Refer to the '`-fprofile-arcs`' option above for a description of *auxname* and instructions on how to generate test coverage data. Coverage data will match the source files more closely, if you do not optimize.

`-g`
`-g`*level* Produce debugging information in the operating system's native format, DWARF 2, and also use *level* to specify how much information. The default level is *0*. Option '`-g`' without a specific level is equivalent to debug option level '`-g2`'.

Open64 allows you to use '`-g`' with '`-O`'. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move to where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

`-g0` Level *0* generates **no** debug information for symbolic debugging.

`-g1` Level *1* produces minimal information, i.e. enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, but no information about local variables and no line numbers. Cuts back on the overhead of full debug information. At this level '`--export-dynamic`' is passed to the linker.

`-g2` Level *2* produces information for symbolic debugging. If **no** optimization options levels are selected, the optimization option level '`-O0`' is invoked to maintain the accuracy of the debugging information. Note the accuracy of the debugging information **cannot** be guaranteed if optimization option levels '`-O1`', '`-O2`', or '`-O3`' are invoked. IPA is **disabled**, if option '`-ipa`' is specified in conjunction with option '`-g2`'.

`-g3` Level 3 includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use '`-g3`'.

`-gdwarf-2`
Produces debugging information in DWARF version 2 format.

`-gdwarf-20`
Produces DWARF 2 debugging information at debug level 0.

`-gdwarf-21`
Produces DWARF 2 debugging information at debug level 1.

`-gdwarf-22`
Produces DWARF 2 debugging information at debug level 2.

`-gdwarf-23`
Produces DWARF 2 debugging information at debug level 3.

`-p`
`-pg` Generate extra code to write profile information suitable for the analysis program `prof`. You must use this option when compiling the source files you want data about, and you must also use it when linking. The '`-p`' option enables application level profiling, see option '`-profile`' to enable library level profiling.

`-profile` Generate extra code to write profile information suitable for the analysis program `gprof`. You must use this option when compiling the source files you want data about, and you must also use it when linking. The '`-profile`' option enables application level and library level profiling.

3.17 Options to Request or Suppress Warnings

Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there may have been an error.

You can request many specific warnings with options beginning '`-W`', for example '`-Wimplicit`' to request warnings on implicit declarations. Some of these specific warning

options also has a negative form beginning '`-Wno-`' to turn off warnings; for example, '`-Wno-implicit`'.

Most of these options have both positive and negative forms; the negative form of '`-Wfoo`' would be '`-Wno-foo`'. This manual typically defines only one of these two forms, whichever one is not the default.

3.17.1 Options that Control Language Independent Warnings

The following options control the amount and kinds of warnings produced by the x86 Open64 compiler for the C/C++ and Fortran Dialects; for further, language-specific options also refer to Section 3.5 [C Dialect Options], page 36 and Section 3.6 [Fortran Dialect Options], page 42.

`-w` Inhibit all warning messages.

`-Wall` Turns on all optional warnings which are desirable for normal code. Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.

`-Wbad-function-cast`
: Warn whenever a function call is cast to a non-matching type. For example, warn if `int malloc()` is cast to `anything *`.

`-Wdeprecated`
`-Wno-deprecated`
: Do not warn about usage of deprecated features.

`-Wdisabled-optimization`
`-Wno-disabled-optimization`
: Warn if a requested optimization pass is disabled. This warning does not generally indicate that there is anything wrong with your code; it merely indicates that the compiler's optimizers were unable to handle the code effectively. Often, the problem is that your code is too big or too complex; GCC will refuse to optimize programs when the optimization itself is likely to take inordinate amounts of time.

`-Wdiv-by-zero`
`-Wno-div-by-zero`
: Do not warn about compile-time integer division by zero. Floating point division by zero is not warned about, as it can be a legitimate way of obtaining infinities and NaNs.

`-Wendif-labels`
`-Wno-endif-labels`
: Do not warn whenever an '`#else`' or an '`#endif`' are followed by text.

`-Werror`
`-Wno-error`
: Make all warnings into errors.

  Make the specified warning into an error. The specifier for a warning is appended, for example '`-Werror=switch`' turns the warnings controlled by

'`-Wswitch`' into errors. This switch takes a negative form, to be used to negate '`-Werror`' for specific warnings, for example '`-Wno-error=switch`' makes '`-Wswitch`' warnings not be errors, even when '`-Werror`' is in effect. You can use the '`-fdiagnostics-show-option`' option to have each controllable warning amended with the option which controls it, to determine what to use with this option.

Note that specifying '`-Werror=`'*foo* automatically implies '`-W`'*foo*. However, '`-Wno-error=`'*foo* does not imply anything.

`-Wfloat-equal`
`-Wno-float-equal`
Warn if floating point values are used in equality comparisons.

`-Wimport`
`-Wno-import`
Inhibit warning messages about the use of the '`#import`' directive.

`-Wlarger-than-`*len*
`-Wno-larger-than-`*len*
Warn whenever an object of larger than *len* bytes is defined.

`-Wno-deprecated-declarations`
Do not warn about uses of functions , variables , and types marked as deprecated by using the `deprecated` attribute.

`-woff` Turn off all named warnings

`-woffall` Turn off all warnings

`-woffoptions`
Turn off all warnings regarding command-line options

`-woff`*num* Instructs the compiler to suppress the specified warning.

For example:

```
opencc -woff2056 -o foo foo.c
```

Instructs the compiler to suppress warning message number 2056.

```
opencc -woff2056-2300 -o foo foo.c
```

Instructs the compiler to suppress warning message numbers 2056 thru 2300.

```
opencc -woff2056-2300,2420-2500 -o foo foo.c
```

Instructs the compiler to suppress warning message number 2056 thru 2300 and 2420 thru 2500.

`-Wundef`
`-Wno-undef`
Warn if an undefined identifier is evaluated in an '`#if`' directive.

`-Wuninitialized`
`-Wno-uninitialized`
Warn if an automatic variable is used without first being initialized or if a variable may be clobbered by a `setjmp` call.

These warnings are possible only in optimizing compilation, because they require data flow information that is computed only when optimizing. If you do

not specify '`-O`', you will not get these warnings. Instead, the compiler will issue a warning about '`-Wuninitialized`' requiring '`-O`'.

If you want to warn about code which uses the uninitialized value of the variable in its own initializer, use the '`-Winit-self`' option.

These warnings occur for individual uninitialized or clobbered elements of structure, union or array variables as well as for variables which are uninitialized or clobbered as a whole. They do not occur for variables or elements declared `volatile`. Because these warnings depend on optimization, the exact variables or elements for which there are warnings will depend on the precise optimization options.

Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed.

These warnings are made optional because the compiler is not smart enough to see all the reasons why the code might be correct despite appearing to have an error. Here is one example of how this can happen:

```
{
  int x;
  switch (y)
    {
    case 1: x = 1;
      break;
    case 2: x = 4;
      break;
    case 3: x = 5;
    }
  foo (x);
}
```

If the value of `y` is always 1, 2 or 3, then `x` is always initialized, but the compiler doesn't know this. Here is another common case:

```
{
  int save_y;
  if (change_y) save_y = y, y = new_y;
  ...
  if (change_y) y = save_y;
}
```

This has no bug because `save_y` is used only if it is set.

This option also warns when a non-volatile automatic variable might be changed by a call to `longjmp`. These warnings as well are possible only in optimizing compilation.

The compiler sees only the calls to `setjmp`. It cannot know where `longjmp` will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact no problem because `longjmp` cannot in fact be called at the place which would cause a problem.

Some spurious warnings can be avoided if you declare all the functions you use that never return as `noreturn`. This warning is enabled by '`-Wall`'.

`-Wunknown-pragmas`
`-Wno-unknown-pragmas`
: Warn when a #pragma directive is encountered which is not understood by the compiler. If this command line option is used, warnings will even be issued for unknown pragmas in system header files. This is not the case if the warnings were only enabled by the '`-Wall`' command line option.

`-Wunreachable-code`
`-Wno-unreachable-code`
: Warn if the compiler detects that code will never be executed.

  This option is intended to warn when the compiler detects that at least a whole line of source code will never be executed, because some condition is never satisfied or because it is after a procedure that never returns.

  It is possible for this option to produce a warning even though there are circumstances under which part of the affected line can be executed, so care should be taken when removing apparently-unreachable code.

  For instance, when a function is inlined, a warning may mean that the line is unreachable in only one inlined copy of the function.

  This option is not made part of '`-Wall`' because in a debugging version of a program there is often substantial code which checks correct functioning of the program and is, hopefully, unreachable because the program does work. Another common use of unreachable code is to provide behavior which is selectable at compile-time.

`-Wunused`
`-Wno-unused`
: Warns whenever a variable is unused. Note in order to get a warning about an unused function parameter, you must specify '`-Wunused-parameter`'.

`-Wunused-function`
`-Wno-unused-function`
: Warn whenever a static function is declared but not defined or a non-inline static function is unused. This warning is enabled by '`-Wall`'.

`-Wunused-label`
`-Wno-unused-label`
: Warn whenever a label is declared but not used. This warning is enabled by '`-Wall`'.

`-Wunused-parameter`
`-Wno-unused-parameter`
: Warn whenever a function parameter is unused aside from its declaration.

`-Wunused-value`
`-Wno-unused-value`
: Warn whenever a statement computes a result that is explicitly not used. This warning is enabled by '`-Wall`'.

`-Wunused-variable`
`-Wno-unused-variable`
: Warn whenever a local variable or non-constant static variable is unused aside from its declaration. This warning is enabled by '`-Wall`'.

`-Wwrite-strings`
`-Wno-write-strings`
: When compiling C, give string constants the type `const char[`*`length`*`]` so that copying the address of one into a non-`const char *` pointer will get a warning; when compiling C++, warn about the deprecated conversion from string literals to `char *`. This warning, by default, is enabled for C++ programs. These warnings will help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using `const` in declarations and prototypes. Otherwise, it will just be a nuisance; this is why we did not make '`-Wall`' request these warnings.

3.17.2 Options that Control C/C++ Warnings

The following options control the amount and kinds of warnings produced by the compiler for the C/C++ dialect only; for further, language-specific options also refer to Section 3.5 [C Dialect Options], page 36.

`-Waggregate-return`
: Warn if any functions that return structures or unions are defined or called. (In C/C++ language where you can return an array, this also elicits a warning.)

`-Wcast-align`
`-Wno-cast-align`
: Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a `char *` is cast to an `int *` on machines where integers can only be accessed at two- or four-byte boundaries.

`-Wchar-subscripts`
`-Wno-char-subscripts`
: Warn if an array subscript has type `char`. This is a common cause of error, as programmers often forget that this type is signed on some machines. This warning is enabled by '`-Wall`'.

`-Wcomment`
`-Wno-comment`
: Warn whenever a comment-start sequence '`/*`' appears in a '`/*`' comment, or whenever a Backslash-Newline appears in a '`//`' comment. This warning is enabled by '`-Wall`'.

`-Wconversion`
`-Wno-conversion`
: Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and conversions changing the width or signedness of a fixed point argument except when the same as the default promotion.

Also, warn if a negative integer constant expression is implicitly converted to an unsigned type. For example, warn about the assignment `x = -1` if `x` is unsigned. But do not warn about explicit casts like `(unsigned) -1`.

`-Wdeclaration-after-statement`
Warn when a declaration is found after a statement in a block. This construct, known from C++, was introduced with ISO C99 and is by default allowed in Open64. It is not supported by ISO C90.

`-Wformat`
`-Wno-format`
Check calls to `printf` and `scanf`, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense. This includes standard functions, and others specified by format attributes , in the `printf`, `scanf`, `strftime` and `strfmon` (an X/Open extension, not in the C standard) families (or other target-specific families). Which functions are checked without format attributes having been specified depends on the standard version selected, and such checks of functions without the attribute specified are disabled by '`-ffreestanding`' or '`-fno-builtin`'.

The formats are checked against the format features supported by GNU libc version 2.2. These include all ISO C90 and C99 features, as well as features from the Single Unix Specification and some BSD and GNU extensions. Other library implementations may not support all these features; The compiler does not support warning about features that go beyond a particular library's limitations. However, if '`-pedantic`' is used with '`-Wformat`', warnings will be given about format features not in the selected standard version (but not for `strfmon` formats, since those are not in any version of the C standard). See Section 3.5 [Options Controlling C/C++ Dialect], page 36.

Since '`-Wformat`' also checks for null format arguments for several functions, '`-Wformat`' also implies '`-Wnonnull`'.

'`-Wformat`' is included in '`-Wall`'. For more control over some aspects of format checking, the options '`-Wformat-y2k`', '`-Wno-format-extra-args`', '`-Wno-format-zero-length`', '`-Wformat-nonliteral`', '`-Wformat-security`', and '`-Wformat=2`' are available, but are not included in '`-Wall`'.

`-Wformat-nonliteral`
`-Wno-format-nonliteral`
If '`-Wformat`' is specified, also warn if the format string is not a string literal and so cannot be checked, unless the format function takes its format arguments as a `va_list`.

`-Wformat-security`
`-Wno-format-security`
If '`-Wformat`' is specified, also warn about uses of format functions that represent possible security problems. At present, this warns about calls to `printf` and `scanf` functions where the format string is not a string literal and there are no format arguments, as in `printf (foo);`. This may be a security hole if the format string came from untrusted input and contains '`%n`'. (This is

currently a subset of what '`-Wformat-nonliteral`' warns about, but in future warnings may be added to '`-Wformat-security`' that are not included in '`-Wformat-nonliteral`'.)

`-wid-clash`
`-wno-id-clash`
Warns if two identifiers have the exact same first `num` characters.

`-Wimplicit`
`-Wno-implicit`
Same as '`-Wimplicit-int`' and '`-Wimplicit-function-declaration`'. This warning is enabled by '`-Wall`'.

`-Wimplicit-function-declaration`
`-Wno-implicit-function-declaration`
Give a warning (or error) whenever a function is used before being declared. The form '`-Wno-error-implicit-function-declaration`' is not supported. This warning is enabled by '`-Wall`' (as a warning, not an error).

`-Wimplicit-int`
`-Wno-implicit-int`
Warn when a declaration does not specify a type. This warning is enabled by '`-Wall`'.

`-Winline`
`-Wno-inline`
Warn if a function can not be inlined and it was declared as inline. Even with this option, the compiler will not warn about failures to inline functions declared in system headers.

The compiler uses a variety of heuristics to determine whether or not to inline a function. For example, the compiler takes into account the size of the function being inlined and the amount of inlining that has already been done in the current function. Therefore, seemingly insignificant changes in the source program can cause the warnings produced by '`-Winline`' to appear or disappear.

`-Wmain`
`-Wno-main`
Warn if the type of '`main`' is suspicious. '`main`' should be a function with external linkage, returning int, taking either zero arguments, two, or three arguments of appropriate types. This warning is enabled by '`-Wall`'.

`-Wmissing-braces`
`-Wno-missing-braces`
Warn if an aggregate or union initializer is not fully bracketed. In the following example, the initializer for '`a`' is not fully bracketed, but that for '`b`' is fully bracketed.

```
int a[2][2] = { 0, 1, 2, 3 };
int b[2][2] = { { 0, 1 }, { 2, 3 } };
```

This warning is enabled by '`-Wall`'.

`-Wmissing-declarations`
`-Wno-missing-declarations`
: Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files.

`-Wmissing-format-attribute`
`-Wno-missing-format-attribute`
: Warn about function pointers which might be candidates for `format` attributes. Note these are only possible candidates, not absolute ones. The compiler will guess that function pointers with `format` attributes that are used in assignment, initialization, parameter passing or return statements should have a corresponding `format` attribute in the resulting type. That is: the left-hand side of the assignment or initialization, the type of the parameter variable, or the return type of the containing function respectively should also have a `format` attribute to avoid the warning.

  The compiler will also warn about function definitions which might be candidates for `format` attributes. Again, these are only possible candidates. The compiler will guess that `format` attributes might be appropriate for any function that calls a function like `vprintf` or `vscanf`, but this might not always be the case, and some functions for which `format` attributes are appropriate may not be detected.

`-Wmissing-noreturn`
`-Wno-missing-noreturn`
: Warn about functions which might be candidates for attribute `noreturn`. Note these are only possible candidates, not absolute ones. Care should be taken to manually verify functions actually do not ever return before adding the `noreturn` attribute, otherwise subtle code generation bugs could be introduced. You will not get a warning for `main` in hosted C environments.

`-Wmissing-prototypes`
`-Wno-missing-prototypes`
: Warn if a global function is defined without a previous prototype declaration. This warning is issued even if the definition itself provides a prototype. The aim is to detect global functions that fail to be declared in header files.

`-Wmultichar`
`-Wno-multichar`
: Do not warn if a multicharacter constant is used. Usually they indicate a typo in the user's code, as they have implementation-defined values, and should not be used in portable code.

`-Wnested-externs`
`-Wno-nested-externs`
: Warn if an `extern` declaration is encountered within a function.

`-Wno-cast-qual`
: Do **not** warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a `const char *` is cast to an ordinary `char *`.

`-Wno-format-extra-args`
: If '`-Wformat`' is specified, do not warn about excess arguments to a `printf` or `scanf` format function. The C standard specifies that such arguments are ignored.

  Where the unused arguments lie between used arguments that are specified with '`$`' operand number specifications, normally warnings are still given, since the implementation could not know what type to pass to `va_arg` to skip the unused arguments. However, in the case of `scanf` formats, this option will suppress the warning if the unused arguments are all pointers, since the Single Unix Specification says that such unused arguments are allowed.

`-Wno-format-y2k`
: If '`-Wformat`' is specified, do **not** warn about `strftime` formats which may yield only a two-digit year.

`-Wlong-long`
`-Wno-long-long`
: Warn if '`long long`' type is used. This is default. To inhibit the warning messages, use '`-Wno-long-long`'. Flags '`-Wlong-long`' and '`-Wno-long-long`' are taken into account only when '`-pedantic`' flag is used.

`-Wnonnull`
: Warn about passing a null pointer for arguments marked as requiring a non-null value by the `nonnull` function attribute.

  '`-Wnonnull`' is included in '`-Wall`' and '`-Wformat`'.

`-Wno-non-template-friend`
: Disable warnings when non-templatized friend functions are declared within a template. Since the advent of explicit template specification support in openCC, if the name of the friend is an unqualified-id (i.e., '`friend foo(int)`'), the C++ language specification demands that the friend declare or define an ordinary, nontemplate function. Before openCC implemented explicit specification, unqualified-ids could be interpreted as a particular specialization of a templatized function. Because this non-conforming behavior is no longer the default behavior for openCC, '`-Wno-non-template-friend`' is off by default allowing the compiler to check existing code for potential trouble spots.

`-Wnon-virtual-dtor`
`-Wno-non-virtual-dtor`
: Warn when a class appears to be polymorphic, thereby requiring a virtual destructor, yet it declares a non-virtual one. This warning is also enabled if -Weffc++ is specified.

`-Wno-pmf-conversions`
: Disable the diagnostic for converting a bound pointer to member function to a plain pointer.

`-Wold-style-cast`
`-Wno-old-style-cast`
: Warn if an old-style (C-style) cast to a non-void type is used within a C++ program. The new-style casts ('`dynamic_cast`', '`static_cast`',

'`reinterpret_cast`', and '`const_cast`') are less vulnerable to unintended effects and much easier to search for.

`-Woverloaded-virtual`
`-Wno-overloaded-virtual`

Warn when a function declaration hides virtual functions from a base class. For example, in:

```
struct A {
  virtual void f();
};

struct B: public A {
  void f(int);
};
```

the `A` class version of `f` is hidden in `B`, and code like:

```
B* b;
b->f();
```

will fail to compile.

`-Wpacked`
`-Wno-packed`

Warn if a structure is given the packed attribute, but the packed attribute has no effect on the layout or size of the structure. Such structures may be misaligned for little benefit. For instance, in this code, the variable `f.x` in `struct bar` will be misaligned even though `struct bar` does not itself have the packed attribute:

```
struct foo {
  int x;
  char a, b, c, d;
} __attribute__((packed));
struct bar {
  char z;
  struct foo f;
};
```

`-Wpadded`
`-Wno-padded`

Warn if padding is included in a structure, either to align an element of the structure or to align the whole structure. Sometimes when this happens it is possible to rearrange the fields of the structure to reduce the padding and so make the structure smaller.

`-Wparentheses`
`-Wno-parentheses`

Warn if parentheses are omitted in certain contexts, such as when there is an assignment in a context where a truth value is expected, or when operators are nested whose precedence people often get confused about. Only the warning for an assignment used as a truth value is supported when compiling C++; the other warnings are only supported when compiling C.

Also warn if a comparison like '`x<=y<=z`' appears; this is equivalent to '`(x<=y ? 1 : 0) <= z`', which is a different interpretation from that of ordinary mathematical notation.

Also warn about constructions where there may be confusion as to which `if` statement an `else` branch belongs. Here is an example of such a case:

```
{
  if (a)
    if (b)
      foo ();
  else
    bar ();
}
```

In C, every `else` branch belongs to the innermost possible `if` statement, which in this example is `if (b)`. This is often not what the programmer expected, as illustrated in the above example by indentation the programmer chose. When there is the potential for this confusion, the compiler will issue a warning when this flag is specified. To eliminate the warning, add explicit braces around the innermost `if` statement so there is no way the `else` could belong to the enclosing `if`. The resulting code would look like this:

```
{
  if (a)
    {
      if (b)
        foo ();
      else
        bar ();
    }
}
```

This warning is enabled by '`-Wall`'.

`-Wpointer-arith`
`-Wno-pointer-arith`
: Warn about anything that depends on the "size of" a function type or of `void`. GNU C assigns these types a size of 1, for convenience in calculations with `void *` pointers and pointers to functions.

`-Wredundant-decls`
`-Wno-redundant-decls`
: Warn if anything is declared more than once in the same scope, even in cases where multiple declaration is valid and changes nothing.

`-Wreorder`
`-Wno-reorder`
: Warn when the order of member initializers given in the code does not match the order in which they must be executed. For instance:

```
struct A {
  int i;
  int j;
  A(): j (0), i (1) { }
};
```

The compiler will rearrange the member initializers for '`i`' and '`j`' to match the declaration order of the members, emitting a warning to that effect. This warning is enabled by '`-Wall`'.

`-Wreturn-type`
`-Wno-return-type`

Warn whenever a function is defined with a return-type that defaults to `int`. Also warn about any `return` statement with no return-value in a function whose return-type is not `void`.

For C, also warn if the return type of a function has a type qualifier such as `const`. Such a type qualifier has no effect, since the value returned by a function is not an lvalue. ISO C prohibits qualified `void` return types on function definitions, so such return types always receive a warning even without this option.

For C++, a function without return type always produces a diagnostic message, even when '`-Wno-return-type`' is specified. The only exceptions are '`main`' and functions defined in system headers.

This warning is enabled by '`-Wall`'.

`-Wsequence-point`
`-Wno-sequence-point`

Warn about code that may have undefined semantics because of violations of sequence point rules in the C and C++ standards.

The C and C++ standards defines the order in which expressions in a C/C++ program are evaluated in terms of *sequence points*, which represent a partial ordering between the execution of parts of the program: those executed before the sequence point, and those executed after it. These occur after the evaluation of a full expression (one which is not part of a larger expression), after the evaluation of the first operand of a `&&`, `||`, `? :` or `,` (comma) operator, before a function is called (but after the evaluation of its arguments and the expression denoting the called function), and in certain other places. Other than as expressed by the sequence point rules, the order of evaluation of subexpressions of an expression is not specified. All these rules describe only a partial order rather than a total order, since, for example, if two functions are called within one expression with no sequence point between them, the order in which the functions are called is not specified. However, the standards committee have ruled that function calls do not overlap.

It is not specified when between sequence points modifications to the values of objects take effect. Programs whose behavior depends on this have undefined behavior; the C and C++ standards specify that "Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.". If a program breaks these rules, the results on any particular implementation are entirely unpredictable.

Examples of code with undefined behavior are `a = a++;`, `a[n] = b[n++]` and `a[i++] = i;`. Some more complicated cases are not diagnosed by this option, and it may give an occasional false positive result, but in general it has been found fairly effective at detecting this sort of problem in programs.

The standard is worded confusingly, therefore there is some debate over the precise meaning of the sequence point rules in subtle cases. Links to discussions

of the problem, including proposed formal definitions, may be found on the GCC readings page, at `http://gcc.gnu.org/readings.html`.

This warning is enabled by '`-Wall`' for C and C++.

`-Wshadow`
`-Wno-shadow`

Warn whenever a local variable shadows another local variable, parameter or global variable or whenever a built-in function is shadowed.

`-Wsign-compare`
`-Wno-sign-compare`

Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. This warning is also enabled by '`-Wextra`'; to get the other warnings of '`-Wextra`' without this warning, use '`-Wextra -Wno-sign-compare`'.

`-Wsign-promo`
`-Wno-sign-promo`

Warn when overload resolution chooses a promotion from unsigned or enumerated type to a signed type, over a conversion to an unsigned type of the same size. Previous versions of openCC would try to preserve unsignedness, but the standard mandates the current behavior.

```
struct A {
  operator int ();
  A& operator = (int);
};

main ()
{
  A a,b;
  a = b;
}
```

In this example, G++ will synthesize a default '`A& operator = (const A&);`', while cfront will use the user-defined '`operator =`'.

`-Wstrict-aliasing`
`-Wno-strict-aliasing`

This option is only active when '`-fstrict-aliasing`' is active. It warns about code which might break the strict aliasing rules that the compiler is using for optimization. The warning does not catch all cases, but does attempt to catch the more common pitfalls. It is included in '`-Wall`'.

`-Wstrict-prototypes`
`-Wnostrict-prototypes`

Warn if a function is declared or defined without specifying the argument types. (An old-style function definition is permitted without a warning if preceded by a declaration which specifies the argument types.)

`-Wswitch`
`-Wno-switch`

Warn whenever a `switch` statement has an index of enumerated type and lacks a `case` for one or more of the named codes of that enumeration. (The presence

of a `default` label prevents this warning.) `case` labels outside the enumeration range also provoke warnings when this option is used. This warning is enabled by '`-Wall`'.

`-Wswitch-default`
Warn whenever a `switch` statement does not have a `default` case.

`-Wswitch-enum`
Warn whenever a `switch` statement has an enumerated type and lacks a `case` for one or more of the named codes of that enumeration.

`-Wsystem-headers`
`-Wno-system-headers`
Print warning messages for constructs found in system header files. Warnings from system headers are normally suppressed, on the assumption that they usually do not indicate real problems and would only make the compiler output harder to read. Using this command line option tells openCC to emit warnings from system headers as if they occurred in user code. However, note that using '`-Wall`' in conjunction with this option will *not* warn about unknown pragmas in system headers—for that, '`-Wunknown-pragmas`' must also be used.

`-Wsynth` (C++ Only)
`-Wno-synth` (C++ Only)
'`-Wsynth`' warns about synthesis that is not backward compatible with cfront. '`-Wno-synth`' suppress warnings about systhesis that is not backward compatible with cfront.

`-Wtraditional`
`-Wno-traditional`
Warn about certain constructs that behave differently in traditional and ISO C. Also warn about ISO C constructs that have no traditional C equivalent, and/or problematic constructs which should be avoided.

- Macro parameters that appear within string literals in the macro body. In traditional C macro replacement takes place within string literals, but does not in ISO C.
- In traditional C, some preprocessor directives did not exist. Traditional preprocessors would only consider a line to be a directive if the '`#`' appeared in column 1 on the line. Therefore '`-Wtraditional`' warns about directives that traditional C understands but would ignore because the '`#`' does not appear as the first character on the line. It also suggests you hide directives like '`#pragma`' not understood by traditional C by indenting them. Some traditional implementations would not recognize '`#elif`', so it suggests avoiding it altogether.
- A function-like macro that appears without arguments.
- The unary plus operator.
- The '`U`' integer constant suffix, or the '`F`' or '`L`' floating point constant suffixes. (Traditional C does support the '`L`' suffix on integer constants.) Note, these suffixes appear in macros defined in the system headers of most modern systems, e.g. the '`_MIN`'/'`_MAX`' macros in `<limits.h>`. Use

of these macros in user code might normally lead to spurious warnings, however the integrated preprocessor has enough context to avoid warning in these cases.

- A function declared external in one block and then used after the end of the block.
- A `switch` statement has an operand of type `long`.
- A non-`static` function declaration follows a `static` one. This construct is not accepted by some traditional C compilers.
- The ISO type of an integer constant has a different width or signedness from its traditional type. This warning is only issued if the base of the constant is ten. I.e. hexadecimal or octal values, which typically represent bit patterns, are not warned about.
- Usage of ISO string concatenation is detected.
- Initialization of automatic aggregates.
- Identifier conflicts with labels. Traditional C lacks a separate namespace for labels.
- Initialization of unions. If the initializer is zero, the warning is omitted. This is done under the assumption that the zero initializer in user code appears conditioned on e.g. `__STDC__` to avoid missing initializer warnings and relies on default initialization to zero in the traditional C case.
- Conversions by prototypes between fixed/floating point values and vice versa. The absence of these prototypes when compiling with traditional C would cause serious problems. This is a subset of the possible conversion warnings, for the full set use '`-Wconversion`'.
- Use of ISO C style function definitions. This warning intentionally is *not* issued for prototype declarations or variadic functions because these ISO C features will appear in your code when using libiberty's traditional C compatibility macros, `PARAMS` and `VPARAMS`. This warning is also bypassed for nested functions because that feature is already an extension and thus not relevant to traditional C compatibility.

`-Wtrigraphs`
`-Wno-trigraphs`

Warn if any trigraphs are encountered that might change the meaning of the program (trigraphs within comments are not warned about). This warning is enabled by '`-Wall`'.

3.18 Environment Variables Affecting x86 Open64

This section describes several environment variables that affect how the compiler operates. System environment variables can be used to affect the behavior of the x86 Open64 compilers. The following lists all such environment variables recognized by the compilers.

3.18.1 Environment Variables for the C/C++ Compiler

`OPEN64_CFLAGS`

Compilation flags to be passed to the C compiler (`opencc`).

`OPEN64_CXXFLAGS`
: Compilation flags to be passed to the C++ compiler (`openCC`).

3.18.2 Environment Variables for the Fortran Compiler

`F90_BOUNDS_CHECK_ABORT`
: Instruct the program to abort upon encountering the first bounds check violation.

`F90_DUMP_MAP`
: Dump memory mapping information when a segmentation fault occurs.

`FTN_SUPPRESS_REPEATS`
: Instead of using the repeat factor, output multiple values at run time.

`NLSPATH` Specify the location of compile-time and run-time error messages. (You can use %N to denote the base name of the file.) This environment variable is useful if the main function of your program is coded in C, and other parts of the program are coded in Fortran. In this case, `NLSPATH` tells the Fortran run time library where to find the file containing the run time error messages.

`OPEN64_FDEBUG_ALLOC`
: Initialize memory locations during program execution. This environment variable is used to debug Fortran memory allocations.

`OPEN64_FFLAGS`
: Compilation flags to be passed to the Fortran compiler (`openf90`, `openf95`).

3.18.3 Language-independent Environment Variables

`FILENV` Specify the location of the assign file.

`HUGETLB_DISABLE_MAPPING_TEXT`
: Specify that huge pages will not be used for text segments. One reason for setting this variable is to allow debugging of code on operating systems where breakpoints cannot be set in huge pages.

`HUGETLB_ELF_LIMIT`
: Specify the maximum number of huge pages to be used for bdt (bss, data, and text segments).

`HUGETLB_LIMIT`
: Specify the maximum number of huge pages to be used for the heap.

`OPEN64_COMPILER_DEFAULTS_PATH`
: Specify a directory or a list of directories to search for the compiler.defaults file.

`OPEN64_GENFLAGS`
: Compilation flags to be passed to all compilers.

`OPEN64_PROBLEM_REPORT_DIR`
: Specify a directory into which the compiler can save preprocessed source files and problem reports in the event the compiler encounters an internal error.

3.18.4 Environment Variables for OpenMP

`OMP_DYNAMIC`
: Specify whether dynamic adjustment of the number of threads available for execution is to be enabled.

`OMP_NESTED`
: Specify whether nested parallelism is to be enabled.

`OMP_NUM_THREADS`
: Specify the number of threads to be used during execution.

`OMP_SCHEDULE`
: Specify the schedule type to be applied to `DO` and `PARALLEL_DO` directives with `RUNTIME` schedule type. `OMP_SCHEDULE` can be any of `STATIC`, `DYNAMIC`, or `GUIDED`.

`OMP_SLAVE_STACK_SIZE`
: Specify the amount of stack size to be used for slave threads.

`O64_OMP_SET_AFFINITY`
: Specify if the operating system's affinity mechanism is used to assign OpenMP threads to CPUs. Assignment of threads to processors occurs if the variable is set to `TRUE` or if the variable is not set. Assignment of threads to processors is disabled if the variable is set to `FALSE`.

`O64_OMP_SPIN_COUNT`
: Specify the number of times the spin loops will spin at user-level before falling back to operating system schedule/reschedule mechanisms. The default value is 20000.

`O64_OMP_SPIN_USER_LOCK`
: Specify whether or not to use user-level spin mechanism for OpenMP locks. If the variable is set to `TRUE` then user-level spin mechanisms are used. If the variable is set to `FALSE` then pthread mutexes are used. The default if the variable is not set is the same as `FALSE`.

`O64_OMP_AFFINITY_MAP`
: Specify the thread-CPU relationship when the operating system's affinity mechanism is used to assign OpenMP threads to CPUs. The variable should contain a list of CPUs. For example, `O64_OMP_AFFINITY_MAP="3 1 2 0"` maps thread 0 to CPU 3, thread 1 to CPU 1, thread 2 to CPU 2 and thread 3 to CPU 0.

4 Binary Compatibility

Binary compatibility encompasses several related concepts:

application binary interface (ABI)
: The set of runtime conventions followed by all of the tools that deal with binary representations of a program, including compilers, assemblers, linkers, and language runtime support. Some ABIs are formal with a written specification, possibly designed by multiple interested parties. Others are simply the way things are actually done by a particular set of tools.

ABI conformance
: A compiler conforms to an ABI if it generates code that follows all of the specifications enumerated by that ABI. A library conforms to an ABI if it is implemented according to that ABI. An application conforms to an ABI if it is built using tools that conform to that ABI and does not contain source code that specifically changes behavior specified by the ABI.

calling conventions
: Calling conventions are a subset of an ABI that specify how arguments are passed and function results are returned.

interoperability
: Different sets of tools are interoperable if they generate files that can be used in the same program. The set of tools includes compilers, assemblers, linkers, libraries, header files, startup files, and debuggers. Binaries produced by different sets of tools are not interoperable unless they implement the same ABI. This applies to different versions of the same tools as well as tools from different vendors.

intercallability
: Whether a function in a binary built by one set of tools can call a function in a binary built by a different set of tools is a subset of interoperability.

implementation-defined features
: Language standards include lists of implementation-defined features whose behavior can vary from one implementation to another. Some of these features are normally covered by a platform's ABI and others are not. The features that are not covered by an ABI generally affect how a program behaves, but not intercallability.

compatibility
: Conformance to the same ABI and the same behavior of implementation-defined features are both relevant for compatibility.

The application binary interface implemented by a C or C++ compiler affects code generation and runtime support for:

- size and alignment of data types
- layout of structured types
- calling conventions

- register usage conventions
- interfaces for runtime arithmetic support
- object file formats

In addition, the application binary interface implemented by a C++ compiler affects code generation and runtime support for:

- name mangling
- exception handling
- invoking constructors and destructors
- layout, alignment, and padding of classes
- layout and alignment of virtual tables

Some x86 Open64 compilation options cause the compiler to generate code that does not conform to the platform's default ABI. Other options cause different program behavior for implementation-defined features that are not covered by an ABI. These options are provided for consistency with GCC and provide compatibility with other compilers that do not follow the platform's default ABI or the usual behavior of implementation-defined features for the platform. Be very careful about using such options.

Most platforms have a well-defined ABI that covers C code, but ABIs that cover C++ functionality are not yet common.

Since the x86 Open64 compiler suite uses the GNU C and C++ front ends (see Chapter 2 [Using the x86 Open64 Compiler], page 9) most binary compatibility issues between various versions of the GNU compiler suite will apply with x86 Open64 (the x86 Open64 compiler suite uses the same version numbering scheme as the GNU suite). Starting with GCC 3.2, GCC binary conventions for C++ are based on a written, vendor-neutral C++ ABI that was designed to be specific to 64-bit Itanium platforms but also includes generic specifications that apply to any platform including x86_64 and i386. Since this ABI is still relatively new, it is possible there can be changes coming from different interpretations of the C++ ABI by different vendors, bugs in the ABI, or bugs in the implementation of the ABI in different compilers.

The C++ library used with a C++ compiler includes the Standard C++ Library, with functionality defined in the C++ Standard, plus language runtime support. The runtime support is included in a C++ ABI, but there is no formal ABI for the Standard C++ Library. Two implementations of that library are interoperable if one follows the de-facto ABI of the other and if they are both built with the same compiler, or with compilers that conform to the same ABI for C++ compiler and runtime support.

When x86 Open64 and another C++ compiler conform to the same C++ ABI, but the implementations of the Standard C++ Library that they normally use do not follow the same ABI for the Standard C++ Library, object files built with those compilers can be used in the same program only if they use the same C++ library. This requires specifying the location of the C++ library header files when invoking the compiler whose usual library is not being used. The location of the x86 Open64 C++ header files depends on how the x86 Open64 build was configured, but can be seen by using the x86 Open64 '`-v`' option. With default configuration options for x86 Open64 4.2 the compile line for a different C++ compiler needs to include

```
-Iopen64_install_directory/include/4.2
```

Similarly, compiling code with x86 Open64 that must use a C++ library other than the GNU C++ library requires specifying the location of the header files for that other library.

The most straightforward way to link a program to use a particular C++ library is to use a C++ driver that specifies that C++ library by default. The `openCC` driver, for example, tells the linker where to find the Open64 C++ library ('`libstdc++`') plus the other libraries and startup files it needs, in the proper order.

If a program must use a different C++ library and it's not possible to do the final link using a C++ driver that uses that library by default, it is necessary to tell `openCC` the location and name of that library. It might also be necessary to specify different startup files and other runtime support libraries, and to suppress the use of the Open64 support libraries with one or more of the options '`-nostdlib`', '`-nostartfiles`', and '`-nodefaultlibs`'.

4.1 Library Compatibility

There are some compatibility issues with libraries compiled with C or other Fortran compilers. The following Fortran compilers have linking object code issues:

- Fortran 90 or 95 compilers implement modules and arrays in vastly different ways.
- Fortran 77 has runtime libraries for I/O and intrinsics that are different, but you can still link both runtime libraries to an executable.
- Fortran g77 has library functions with the same names as x86 Open64, but some of the calling conventions are different.

4.1.1 Linking

As described in Section 2.3 [Mixed Code], page 18, for large applications of Fortran and C/C++ code where the main entry to your application is from C or C++, you can optionally use `opencc` or `openCC` to link the application instead of `openf95`. To link object files that were generated with `opencc` or `openCC` include the option '`-lstdc++`'.

If you use `opencc`, when calling libm functions, include this option to the link line, '`-lm`'. The compiler may require an explicit '`-lm`' for the second pass of feedback compilation.

4.1.2 Name Mangling

Name mangling is a way to ensure that function, subroutine, and common-block names from a Fortran program or library are unique and do not conflict with names in libraries from other programming languages. Name mangling also provides a way for the compiler to pass additional information to the linker. Since similar functions in different libraries have the same name, name mangling prevents name clashing when mixing code from C, C++ and Fortran. For example, both the Fortran library and the standard C library have a function named "access". The Fortran library `access` function takes four arguments, while the standard C library function takes only two arguments. Mangling the Fortran symbols prevents a symbol name clash.

By default, the x86 compiler uses the same name mangling conventions as the GNU g77 compiler and libf2c library. Names without an underscore have a single underscore appended (for example, Fortran subroutine called `foo` becomes `foo_`. Names containing an underscore have two underscores appended (for example, `run_check` becomes `run_check__`).

The following options can change this behavior:

'`-fno-underscoring`'
: Instructs the compiler not to append underscores to symbols.

'`-fno-second-underscore`'
: Instructs the compiler not to append a second underscore to a symbol that already contains an underscore. '`-fno-underscore`'.

The compiler also performs name mangling on common block names. For the blank common block, the x86 Open64 uses `_BLNK__` which is the same name used by GNU g77. However, some compilers use `_BLANK__`.

4.1.3 ABI Compatibility

The x86 Open64 supports the official x86_64 Application Binary Interface (ABI), with exception of so called red zone support. The red zone is a 128-byte area beyond the location of the stack pointer that will not be modified by signal handlers or interrupt handlers and therefore can be used for temporary data without adjusting the stack pointer. Lack of red zone support does not affect mixing of Open64 code with gcc code.

The g77 compiler does not support the official x86_64 Application Binary Interface (ABI). Specifically, the g77 does not pass the return values from functions returning `COMPLEX` or `REAL` values as stated in the x86_64 ABI standard.

Character strings passed to subprograms are represented with a character pointer and the integer length parameter is added to the end of the call list.

4.1.3.1 Linking with g77-compiled Libraries

To link with a library compiled by g77 and that library contains functions that return `COMPLEX` or `REAL` types, you need to:

- identify the functions returning `COMPLEX` or `REAL` values
- instruct the compiler (using the '`-ff2c-abi`' switch) that when it generates code calling these functions, it modifies its ABI behavior to match what is expected by g77.

You can only use the '`-ff2c-abi`' switch once on the command line. If you have more than one g77-compiled library, place all the appropriate symbol names into a single file. The format of this file is one symbol per line. Do not mangle the symbol names; specify them as you would in your Fortran code.

The following typescript contains an example, where lib.a contains g77-compiled functions foo and bar.

```
$ cat list
   foo
   bar
   $ openf90 -ff2c-abi list prog.f lib.a
```

4.1.3.2 AMD Core Math Library (ACML)

ACML provides a free set of thoroughly optimized and threaded math routines for HPC, scientific, engineering and related compute-intensive applications. Complete information on ACML is available at the following AMD website:

`http://developer.amd.com/acml.aspx`

4.2 GNU Compatibility

The x86 Open64 is compatible with gcc and g77. Many packages will check that you are using gcc (for example, checking strings like the gcc version).

Some compatibility issues to be aware of:

- Some packages may use deprecated features of gcc. If they do, x86 Open64 will probably print an error and exit, while the gcc will print a warning and continue. For example, some packages still use the deprecated '`-Xlinker`' gcc flag to pass arguments to the linker, while the x86 Open64 uses the '`-W1`' flag.
- Not all gcc flags are implemented yet, but will be later documented in release notes.
- Some packages may have the requirement to call the compiler used "gcc" in order to build correctly.

4.3 Compatibility with Other Fortran Compilers

The x86 Open64 accepts the same source code of other compilers provided that the program conforms strictly to the Fortran 95 standard. The compiler is compatible with g77 with a few exceptions (such as `kind=` type parameters) and even if a program uses extensions (such as additional intrinsic functions).

However, when it comes to linking object files generated by two different compilers, x86 Open64 is generally not compatible with other Fortran compilers (such as gfortran, g95, or commercial compilers), especially if the source code uses language features beyond Fortran 77. The x86 Open64 is compatible with g77, but you must use the command line option '`-ff2c-abi`'. This option handles situations where g77 deviates from the Linux standard ABI for the x86-64 machine. Some issues affecting linking compatibility are due to compilers using different:

- Application Binary Interface (ABI) and data representation.
- runtime libraries to perform I/O, string manipulation, and operations that are too difficult to perform in line. In contrast with the C language where the standard dictates that the runtime library provide functions named `strcpy`, `strcmp`, and `fputs` to copy, compare, and write strings, the Fortran standard describes the behavior of assignment using "=" , operators like ".ge.", and statements like "write" and "format". It is the implementation that chooses names for any runtime library function used to implement that behavior.
- data structures (called "dope vector") to implement an assumed-shape array argument, allocatable array, or Fortran pointer.
- strategy to "mangle" or "decorate" module level identifiers to generate symbols which will not clash in the namespace of the linker.
- strategy to implement the `use` statement.

You may still be able to link object files generated by compilers other than g77 or if the program uses Fortran 90 features and later standards, by following these suggestions:

- For code generated by one compiler that calls a procedure generated by another:
  - Use Fortran 77 style of procedure call.

  - Avoid any dummy arguments requiring the calls to be "explicit" in Fortran 90 and later standards.
  - Do not `use` a module generated by one compiler in a procedure generated by another.
- For name-mangling problems in Fortran 77 style external identifiers, use '`-fno-second-underscore`' and '`-fdecorate`' as needed. For the g77, use '`-fsecond-underscore`' instead.
- When linking with one compiler, specify explicitly the additional runtime libraries needed by the other compiler. For more control over the order the linker scans libraries, run the linker directly. You need to specify the startup object used by the first compiler and the union of the sets of libraries used by the two compilers. To print the names of these objects and libraries, run `openf95` with the command-line option '`-show`'.
- Perform all I/O in code generated by one compiler whenever possible. If not, ensure that all I/O related to a specific logical unit and file occurs within code generated by one computer.

4.4 Porting

If you are porting Fortran code, some considerations are:

- Intrinsics. The x86 Open64 supports most intrinsics.
- Name mangling. See Section 4.1.2 [Name Mangling], page 119.
- Static Data. If your code expects data to be initialized to zero and allocated in the heap, use '`-static`' flag when compiling.

If you are porting existing code to x86-64, consider the following:

- Some source packages assume the locations of libraries and fail to look in `lib64-named` directories for libraries. This may result in unresolved symbols during the link.
- For the x86 platform, specify by using: '`-mcpu=x86_64`'.

4.5 Procedure to Migrate from Other Compilers

Follow this procedure to migrate code from other compilers to the x86 Open64.

1. In your makefile, check the compiler name to ensure that the correct compiler is being called. For example, you may need to add:

   ```
   $ CC=opencc  ./configure <options>
   ```

   In your makefile, change the compiler to `opencc` or `openf95`.
2. Ensure that options called are supported. See Section 3.1 [x86 Open64 Option Summary], page 27, for a complete listing of options.
3. If you use Inter-Procedural Analysis (IPA), see Section 3.9.4 [Options that Control Interprocedural Optimizations], page 63 for suggestions.
4. Try compiling your code and examining the results. Look for missing libraries that were previously linked automatically and for behavior differences.

5 Tuning Applications Using the x86 Open64 Compiler Suite

Now that you know the basics and have compiled a few programs using the x86 Open64 compiler, you are ready to try new methods to optimize the performance and increase the speed of your applications. This chapter describes in depth how to use various tuning options, OpenMP and autoparallelization to achieve this goal.

5.1 Global Optimizations

The '`-O`' flag specifies the level of optimization. The different levels are:

- Flag '`-O0`' means no optimization. This level is used when the '`-g`' flag is set for debugging.
- Flag '`-O1`' specifies local optimizations on sections of straight-line code (basic blocks) only. Examples of such optimizations are instruction scheduling and some peephole optimizations. These optimizations do not usually have any noticeable impact on compilation time.
- Flag '`-O2`' specifies additional global optimizations. Examples of such optimizations are control flow optimizations, partial redundancy elimination and strength reduction. These optimizations can often significantly reduce the execution time of the compiled program, but may do so at the expense of increased compilation time. The '`-O2`' option is the default. Optimizations at '`-O2`' include:
  - Loop unrolling, simple if-conversion and recurrence-related optimizations in inner loops.
  - Two pass instruction scheduling with global register allocation after the first pass.
  - Global optimizations within function scopes: partial redundancy elimination, strength reduction and loop termination test replacement, dead store elimination, control flow optimizations, and instruction scheduling across basic blocks.
  - Enabling the conversion of GOTOs into higher level structures like FOR loops.
  - Set '`OPT:Olimit=6000`'
- Flag '`-O3`' includes all '`-O1`' and '`-O2`' optimizations and additional more aggressive optimizations. Examples of such aggressive optimizations are loop nest optimizations and generation of prefetch instructions. Although these more aggressive optimizations can significantly speed up run time execution of the compiled program, in rare cases they may not be profitable and may instead lead to a slow down. Some of these more aggressive optimizations may also affect accuracy of floating point computations. Optimizations at '`-O3`' include:

  '`-LNO:opt=1`'
  : This flag turns on Loop Nest Optimization

  '`-OPT`'
  : Use with the following options:
    - OPT:roundoff=1
    - OPT:IEEE_arith=2
    - OPT:Olimit=9000
    - OPT:reorg_common=1

In cases where '`-O3`' slows your program, try using `-O3 -LNO:prefetch=0` especially for codes that fit in cache.

5.2 Inter-Procedural Analysis (IPA)

Software applications are usually organized into multiple source files. During the compilation process, the *Makefile* instructs the compiler to compile each source file (compilation unit) separately. This type of build process is called *separate compilation*. Once all compilation units have been compiled into .o files, the linker is invoked and produces the final executable.

Using separate compilation has the drawback of not providing the compiler with complete program information. Where the program accesses external data or calls external functions, the compiler has to make worst-case assumptions. Unknown information, for instance on external function calls or on external data accesses, may reduce the effectiveness of compiler optimization. Open64 supports whole program optimization using interprocedural analysis (IPA). Whole program optimizations enable additional optimizations.

The following sections describe IPA:

- Compilation model
- Analysis and optimizations
- Improvements to backend optimization
- Related flags

5.2.1 IPA Compilation Model

Inter-procedural compilation is the means by which whole program compilation is enabled. IPA requires a different compilation model than separate compilation. By specifying the '`-ipa`' flag, this new mode of compilation is invoked.

For whole program compilation, the compiler must have the entire program for analysis and optimization. This is achieved only after a link step is applied. Usually after all optimization and code generation is performed, the link step is applied to .o files. However, in the IPA compilation model, the link step is applied before most optimization and code generation. In these cases, the program code being linked are in the form of the intermediate representation (IR) used during compilation and optimization, not in the object code format. Only when the program is linked at the IR level are the inter-procedural analysis and optimizations applied to the whole program. Compilation then continues with the backend phases to generate the final object code.

Ease-of-use is one of the main objectives in design of the IPA compilation model. The user only needs to add the '`-ipa`' flag to both the compile line and the link line. There is no need to re-structure your Makefiles to use IPA. To allow this, IPA includes a new kind of .o file that we call IPA .o's. These files are different from the .o files used in separate compilation, which contain object code. In the IPA .o files, program code is in the form of IR. To produce IPA .o files, compile each file with the flags '`-ipa -c`'. Only the IPA linker can link IPA .o files. Adding the '`-ipa`' flag to the link command, invokes the IPA linker. The IPA link step performs the following actions:

1. calls the IPA linker
2. performs inter-procedural analysis and optimization on the linked program

3. calls the backend phases to optimize and generate the object code
4. calls the object code linker to produce the final executable.

Using IPA compilation, the compilation of separate files proceeds very quickly because it does not involve the backend phases. However, the linking phase appears much slower because it now performs compilation and optimization of the entire program.

5.2.2 IPA Analysis

During the IPA analysis phase, an analysis is performed to collect information over the entire program. First, IPA constructs the program call graph. For each function in the program, there is a corresponding node in the call graph. The call graph represents the calling relationships in the program.

The call graph may be rebuilt based on the different inlining heuristics. Once the call graph construction is completed, IPA performs inline analysis to obtain a list of function calls.

IPA also computes alias information for all program variables. A variable may be pointed to by a pointer when its address is taken. Code that dereferences or stores through the pointer may possibly access the variable. IPA's alias analysis keeps track of this information. Despite pointer accesses, IPA alias computation works to identify as few aliases for variables as possible.

5.2.3 IPA Optimization

Inlining is the most important optimization performed by IPA. For inlining, a call to a function is replaced by the actual body of the function which eliminates function call overhead. Since all user function definitions are visible, inlining in IPA is very versatile. Inlining increases the optimization opportunities of the backend phases by enabling them to work on larger pieces of code. For example, the creation of a loop nest that enables aggressive loop transformation may result from inlining.

You should do a benefit analysis of inlining because overuse may result in degraded performance. Some considerations are:

- Increased program size may cause a higher instruction cache miss rate.
- Functions that are already large may result in the compiler running out of registers, so it has to use memory more often causing the program to slow down.
- Too much inlining can slow the later phases of the compilation process.

Function calls often pass constants (including variable addresses) as parameters. Replacing a formal parameter by its constant value helps in optimizing the function body. In many cases, part of the function code can be determined to be useless and deleted. Another way to increase optimization is to use *function cloning* which creates different clones of a function with its parameters customized to the forms of the calls. Function cloning provides some of the benefits of inlining without increasing the size of the function that contains the call. It does, like inlining, increase the total size of the program.

For all calls that pass the same constant parameter, IPA will perform constant propagation for that parameter. This has the same benefit as function cloning but without the increase in program size. IPA will also applies constant propagation to global variables. If

IPA determines that a global variable is constant throughout the entire program execution, it replaces the variable by the constant value.

Dead variable elimination finds and deletes never-used global variables. The IPA often finds these variables when it performs constant propagation.

Dead function elimination finds and deletes never-called functions. These functions may be the by-product of inlining and cloning.

Common padding is applicable to the common blocks in Fortran programs. Usually, compilers can't change the layout of the user variables in a common block because this would require coordination between the different subroutines using the same common block and the subroutines may belong to different compilation units. However, under IPA, all subroutines are available. The padding improves the alignments of the arrays. This mean the arrays can be accessed more efficiently and even vectorized. Also, the padding can reduce data cache conflicts during execution.

Common block splitting is applicable to the common blocks in Fortran programs. Common blocks are split into a number of smaller blocks which reduces data cache conflicts during execution.

Procedure reordering places the program functions in an order based on their call relationship. This reordering may reduce thrashing in instruction cache during execution.

5.2.4 IPA Controls

It is not always possible for the compiler to make the best choices regarding how to optimize a program. The compiler instead provides the user with many compilation options to tune their program for peak performance. IPA is one of the compilation phases that can benefit substantially from feedback compilation. In feedback compilation, the compiler is presented with a feedback data file which contains a profile of a typical run of the program. With this file, the IPA can make better decisions about what functions to inline and clone. For example, the IPA can determine and place busy callers and callees next to each other. Procedure reordering will also be more effective. You enable feedback compilation by using the following options: '`-fb-create`' and '`-fb-opt`'.

5.2.4.1 Inlining

The x86 Open64 has two kinds of inliners, depending on whether '`-ipa`' is specified. This is due to the fact that nowadays inlining is a language feature and must be performed independent of IPA. When '`-ipa`' is not specified, the inliner invoked is the lightweight inliner that can operate only on a single compilation unit. The lightweight inliner does not do automatic inlining, but inlines strictly according to the C++ language requirement, C inline keyword or any user-specified '`-INLINE`' options. The lightweight inliner may be invoked by default. The options for the lightweight inliner are:

'`-INLINE or -inline`'
: Invokes the lightweight inliner when '`-ipa`' is not specified.

'`-INLINE:=off`'
Suppresses the invoking of the lightweight inliner.

The following options are applicable to both the lightweight and IPA's inliner:

'`-INLINE:all`'
Performs all possible inlining. Use this option only if your program is small because it could result in code bloat.

'`-INLINE:list=ON`'
The inliner will list its actions on the fly. Use this option to find out which functions are being inlined and which functions are not being inlined and why. Based on the reasons specified by the output of this flag, you can tweak the inlining controls to inline or not inline a function.

'`-INLINE:must=name1[ ,name2, . . ]`'
To force inlining for the named functions.

'`-INLINE:never=name1[ ,name2, . . ]`'
To suppress inlining for the named functions.

When '`-ipa`' is specified, IPA will invoke its own inliner. In addition to the functions that are required to be inlined, the IPA's inliner automatically determines additional functions. The IPA's inliner has the following preferences:

- Small callees or callers over larger ones
- If profile data is available, calls executed frequently.
- If profile data is not available, calls inside loops.
- Leaf routines (functions containing no calls)

Inlining continues until there are no more calls that satisfy the inlining criteria. The following options control inlining:

'`-IPA:inline=OFF`'
Turns off the IPA's inliner. Since IPA is invoked, the lightweight inliner is also suppressed. The default is ON.

'`-INLINE:none`'
Turns off automatic inlining by IPA. The IPA still performs the inlining required by the language or specified by the user. By default, automatic inlining is turned ON.

'`IPA:specfile=filename`'
Directs the compiler to open the specified file to read more '`-IPA:`' or '`-INLINE:`' options.

Use the following options to tune the aggressiveness of the inliner. Keep in mind that very aggressive inlining can cause performance degradation.

'`-OPT:Olimit=N`'
Specifies the size limit N, where N is computed from the number of basic blocks that make up a function. N=0 means that no limit is imposed. Note that inlining will never cause a function to exceed the size limit. The default under '`-O2`' is 6000. The default under '`-O3`' is 9000.

‘`-IPA:space=N`’
Specifies that inlining should continue until it reaches a factor of N% increase in code size. The default is 100%. This value can be increased if the program size is small.

‘`-IPA:plimit=N`’
Suppresses inlining into a function once its size reaches N, where N is based on the number of basic blocks and the number of calls inside a function. The default is 2500.

‘`-IPA:small_pu=N`’
Specifies that a function with size smaller than N, where N is based on the number of basic blocks is not subject to ‘`-IPA:plimit`’. The default is 30.

‘`-IPA:callee_limit=N`’
Specifies that a function whose size exceeds this limit will never be automatically inlined by IPA. The default is 500.

‘`-IPA:min hotness=N`’
Specifies the “hotness” of a function before it can be inlined by IPA. The “hotness” of a function is directly proportional to the call frequency at the call site and inversely proportional to the ratio of callee size to program size. This option is applicable only under feedback compilation.

‘`-INLINE:aggressive=ON`’
Increases the aggressiveness of the inlining, allowing more non-leaf and out-of-loop calls to be inlined. The default is OFF.

As previously stated, leaf functions are good candidates to be inlined because they do not contain calls that may inhibit various backend optimizations. To increase the effectiveness of leaf functions, IPA provides two options that exploit its call-tree-based inlining feature. This is due to the fact that a function that calls only leaf functions can become a leaf function if all of its calls are inlined. This can be applied repeatedly up the call graph. In the following option descriptions, a function is said to be at *depth N* if it is never more than N edges from a leaf node in the call graph. A leaf function has depth=0.

‘`-IPA:maxdepth=N`’
IPA inlines all routines at depth N in the call graph subject to space limitation.

‘`IPA:forcedepth=N`’
IPA inlines all routines at depth N in the call graph regardless of space limitation.

5.2.5 Cloning

The options for controlling cloning are:

‘`-IPA:multi_clone=N`’
Specifies the maximum number of clones that can be created from a single function. The default is 0 (cloning is turned OFF by default).

‘`-IPA:node_bloat=N`’
Specifies the maximum percentage growth in the number of procedures relative to the original program that cloning can produce. The default is 100.

5.2.6 Additional IPA Tuning Options

The following options are useful in tuning, but are unrelated to inlining and cloning:

'`-IPA:common_pad_size=N`'
: Specifies that common block padding should use pad size of up to N bytes. The default is 0, which means that the compiler determines the best padding size.

'`-IPA:linear=ON`'
: Enables linearization of array references. When inlining Fortran subroutines, IPA attempts to map formal array parameters to the shape of the actual parameters. When this option is ON, the IPA performs inlining but linearizes the array references. These linearizations may degrade the performance, but the inlining may produce more performance gains.

'`-IPA:pu_reorder=`*N*'
: Controls IPA's procedure reordering optimization where N is:

  0 disables the optimization

  1 enables reordering based on the frequency in which different procedures are invoked.

  2 enables procedure reordering based on the caller-callee relationship.

  The default is N=0.

'`-IPA:field_reorder=ON`'
: Enables IPA's field reordering optimization to minimize data cache misses. This optimization is based on reference patterns of fields in large structs which were discovered during feedback compilation. The default is OFF.

'`-IPA:ctype=ON`'
: Optimizes interfaces to constructs defined in the standard header file '`ctype.h`' by assuming that the program is not running in a multi-threaded environment. The default is OFF.

5.2.7 Disabling Options

The following options disable various optimizations in IPA. Use these options to study the effects of optimizations.

'`-IPA:alias=OFF`'
: Disables IPA's alias and mod-ref analysis

'`-IPA:addressing=OFF`'
: Disables IPA's address-taken analysis (a component of the alias analysis)

'`-IPA:cgi=OFF`'
: Disables the constant propagation for global variables (constant global identification)

'`-IPA:cprop=OFF`'
: Disables the constant propagation for parameters

'`-IPA:dfe=OFF`'
: Disables dead function elimination

'`-IPA:dve=OFF`'
Disables dead variable elimination

'`-IPA:split=OFF`'
Disables common block splitting

5.2.8 Invoking IPA

You can invoke inter-procedural analysis with the following: '`-ipa`', '`-IPA`', and implicitly by '`-Ofast`', which turns on '`-ipa`' as part of its optimizations. IPA can be used with any optimization level, but the greatest benefit comes when combined with '`-O3`'.

As previously stated, the .o files created when compiling with '`-ipa`' are not regular .o files. IPA uses these .o files in its analysis of your program. Then the IPA performs a second compilation using the information it learned from the analysis to optimize the executable.

The IPA linker checks that the entire program is compiled with the same set of optimization options. If different optimization options are used, IPA warns:

```
Warning: Inconsistent optimization options detected between files
involved in interprocedural optimization.  Optimization will be honored
for the functions in the file.
```

In the following example, the IPA gives the above warning for C files '`a.c`' and '`b.c`'.

```
~ $ opencc -O2 -ipa -c a.c
~ $ opencc -O3 -ipa -c b.c
~ $ opencc -ipa a.o b.o
```

To remove the warning, the user can pass consistent optimization options to the individual compilations. In the above example, the user can pass '`-O2`' or pass '`-O3`' to both files.

The following example shows a command line using IPA when there are only a few source files:

```
openf95 -O3 -ipa main.f foo1.f foo2.f
```

If you compile files separately, the generated *.o files do not contain object code, but a representation of the source code. The actual compilation occurs at link time. You would also need to add the '`-ipa`' flag to the link command as shown in the following example:

```
openf95 -c -O3 -ipa main.f
openf95 -c -O3 -ipa foo1.f
openf95 -c -O3 -ipa foo2.f
openf95 -O3 -ipa main.o foo1.o foo2.o
```

Note that currently there is a restriction that each archive (for example, '`libfoo.a`') must contain either .o files compiled with '`-ipa`' or .o files compiled without '`-ipa`', but not both.

In a non-IPA compile, compiling all files to create the object files is time consuming, but the link step is quite fast. In an IPA compile, creating the .o files is very fast, but the link step takes time. The total compile time can be significantly longer with IPA than without it.

When invoking the final link phase with '`-ipa`', a large part of this process can be done in parallel on a system with multiple processing units. This feature can be invoked by using the '`-IPA:max_jobs`' flag. This flag has the following options:

'`-IPA:max_jobs=N`'
Limits the maximum parallelism when invoking the compiler after IPA to (at most) N compilations running at once. The values for N are:

0 means the parallelism selected is equal to either the number of processors, the number of cores, or the number of hyperthreading units in the compiling system, whichever is larger.

1 disables parallelization during compilation

>1 specifically sets the degree of parallelism

The default is N=1.

5.3 Loop Nest Optimization (LNO)

Use the Loop Nest Optimization group of flags on programs containing many nests of loops. This group defines transformations and options applicable to loop nests. The LNO feature is invoked by default at '`-O3`'. On certain matrix operations at '`-O3`', LNO can provide a 10 to 20 times performance advantage over other compilers.

In rare cases where this feature can slow things down, you can try using '`-LNO:opt=0`' to disable nearly all loop nest optimizations. You cannot make an '`-O2`' compile faster by adding '`-LNO:opt=0`' because the '`-LNO`' feature is only active with '`-O3`' (or '`-Ofast`' which implies '`-O3`').

With the LNO group of flags, you can control:

- Loop fusion and fission
- Blocking to optimize cache line reuse
- Cache management
- Translation Lookaside Buffer (TLB) optimizations
- Prefetch

5.3.1 Loop Fusion and Fission

Loop fusion is when loop nests that have too few instructions and consecutive loops are combined to improve usage of processor's resources. Loop fission is the opposite when loops are split up because loop nests have too many instructions, or deal with too many data items in their inner loop. This scenario may lead to too much pressure on the registers, resulting in spills of registers to memory. The LNO options to control these transformations are:

'`-LNO:fusion=N`'
Performs loop fusion, where N is:
0 - OFF
1 - Conservative
2 - Aggressive. The compiler attempts to fused the outer loops in consecutive

loop nests, even if analysis decides that fusing all levels of the loop is not beneficial.

The default level is 1 (standard outer loop fusion).

'`-LNO:fission=N`'
Performs loop fisson, where N is:
0 - OFF
1 - Standard
2 - Instructs compiler to try fission before fusion. Level 2 has proved to be beneficial to a number of codes.

The default level is 0.

'`-LNO:fusion_peeling_limit=N`'
Controls the limit for the number of iterations allowed to be peeled during fusion. The default is N=5, but N can be any non-negative integer.

The compiler performs peeling when the iteration count in consecutive loops is close, but different, and when the loop counts are the same because several iterations are replicated outside the loop body.

Use caution when mixing '`-LNO:fusion`' and '`-LNO:fission`' because fusion has precedence over fission. For example, if '`-LNO:fission= [1 or 2]`' and '`-LNO:fussion= [1 or 2]`', then fusion is performed.

5.3.2 Cache Size

For the following AMD processors, the L2 cache sizes are:

1 Mbyte - AMD Opteron^{TM} processor (default)
512 Kbytes or 1 Mbyte - AMD Athlon^{TM} processor

If your target machine has an AMD Athlon^{TM} processor that has a smaller cache size, it may help to set '`-LNO:cs2=512k`'. Alternately, you can specify your target machine by using '`-march=athlon 64`' - this will automatically set the standard machine cache sizes.

The cache size options are:

'`-LNO:cs1=n, cs2=n, cs3=n, cs4=n`'
This option specifies the cache size, where n can be 0 or a positive integer followed by k or K (Kbytes) or m or M (Mbytes). Setting n to 0 means there is no cache at that level.

The types of cache are: cs1 - primary cache
cs2 - secondary cache
cs3 - memory
cs4 - disk

The default cache sizes depend on your system. To list the default cache sizes used during compilation, use '`-LIST:options=ON`'.

'`-LNO:assoc1=n, assoc2=n, assoc3=n, assoc4=n`'
This option specifies the cache set associativity. For a small cache size, the cache set associativity is usually decreased as well.

5.3.3 Cache Blocking, Loop Unrolling, and Interchange Transformations

Cache blocking (tiling) is the process of choosing the appropriate loop interchanges and loop unrolling sizes at the correct levels of the loop nest. This process is done to optimize cache reuse and reduce memory accesses. The default is for this whole LNO feature to be ON. Some cache blocking options are:

'`-LNO:blocking=off`'
Turns off cache blocking.

'`-LNO:blocking_size=N`'
Specifies a block size that the compiler must use when performing any blocking. The value for N is a positive integer that represents the number of iterations.

'`-LNO:interchange`'
Disables the loop interchange transformation in the loop nest optimizer if set to =0. The default is ON.

The -LNO group controls outer loop unrolling. The -OPT group controls inner loop unrolling. The options for controlling loop unrolling are:

'`-LNO:outer_unroll_max, ou_max=N`'
Specifies that the compiler may unroll outer loops in a loop nest by up to N per loop, but no more. The default is N=10.

'`-LNO:ou_prod_max=N`'
Indicates that the product of unrolling levels of the outer loops in a given loop nest is not to exceed N (a positive integer). The default is N=16.

'`-LNO:outer_unroll, ou=N`'
Indicates that exactly N outer loop iterations be unrolled, if unrolling is legal. Unrolling is not done for loops where outer unrolling is not advisable.

5.3.4 Prefetch

Use the following LNO options to provide guidance to the compiler about the level and type of prefetching to enable:

'`-LNO:prefetch=N`'
Specifies how aggressively to prefetch, where N is:
0 - Disables prefetching in loop nests
1 - Default
2 - Prefetch more aggressively than the default
3 - Prefetch even more aggressively than '`-LNO:prefetch=2`'

'`-LNO:prefetch_ahead=N`'
Defines how many cache lines ahead of the current data being loaded should be prefetched. The default is N=2.

5.3.5 Vectorization

Vectorization is an optimization technique that enables the compiler to perform multiple operations at once and can greatly increase speed. For example, the compiler will turn the function `sin()` into a call to `vsin()`, which is twice as fast. Use the following flags for vectorization:

'`-LNO:vintr=N`'
: Controls the use of vectorized versions of functions in the math library, where n is:
0 - Turns OFF vectorization of math intrinsics
1 - Turns ON vectorization of math intrinsics
2 - Vectorizes all math functions. This may be unsafe because the vector forms of some functions may have accuracy problems.

This default is N=1.

'`-LNO:simd=N`'
: Enables or disables inner loop vectorization, where n is:
0 - Turns off the vectorizer
1 - Instructs the compiler to vectorize only if it can determine that there is no unacceptable impact on performance due to sub-optimal alignment.
2 - Vectorizes without any constraints. This is the most aggressive setting.

The default is N=1.

'`-LNO:simd_verbose=ON`'
: Prints vectorized information to stdout.

'`-LNO:vintr_verbose=ON`'
: Prints information on whether or not the math intrinsic functions were vectorized.

5.4 Code Generation ('`-CG`')

The '`-CG`' group controls some parts of instruction-level code generation that can benefit code tuning. The code generation options are:

'`-CG:gcm=OFF`'
: Turns off the instruction-level global code motion optimization phase. The default is ON.

'`-CG:load_exe=N`'
: Specifies the threshold for subsuming a memory load operation into the operand of an artihmetic instruction, where N is:
0 - Turns off this subsumption optimization
1 - By default the compiler performs this subsumption only when the result of the load has only one (N=1) use.

If the number of times the result of the load is used exceeds the value N, then this subsumption is not performed. Setting '`load_exe=2 or 0`' is sometimes

beneficial. For 64-bit ABI and Fortran, the default is N=2; otherwise the default is N=1.

'`-CG:use_prefetchnta=ON`'
Instructs the compiler to use the prefetch operation that assumes that data is Non-Temporal at All (NTA) levels of the cache hierarchy. This option is for data streaming situations where the data will not be reused soon. The default is OFF.

5.5 Feedback Directed Optimization (FDO)

Feedback directed optimization collects profile information about the program by using a special instrumented executable. For example, FDO records the frequency of when every if() statement is true. This information is used in later compilations in tuning the executable. This feature is invoked with the '`-fb-create`' and '`-fb-opt`' flags.

FDO is most helpful when a program's typical execution is somewhat similar to the execution of the instrumented program on its input data set. For example, in cases where the if() frequency of one input data set is significantly different from another set of data, then using FDO may actually slow down the program.

The following procedure shows how to invoke the FDO feature. First, you should note that if '`-fb-create`' and '`-fb-opt`' compiles are done with different compilation flags, FDO may not work. It depends on whether the different compilation flags cause different code to be seen by the phase that is performing the instrumentation/feedback. It is recommended to use the same flags for both instrumentation and feedback.

1. For FDO, you need to compile the program twice in two passes. For the first pass, use this command line:
   ```
   opencc -O3 -ipa -fb-create mydata -o myexe mycode.c
   ```
   The executable '`myexe`' will contains extra instrumentation library calls to collect feedback information. So the downside is '`myexe`' will run slower than normal.
2. Run the program '`myexe`' with a sample data set:
   ```
   ./myexe <input_data>
   ```
   During this run, a file with the prefix of the file name you used will be created, containing feedback information. In the example, the output file might be named '`mydata.instr1.bc123`'. Each file is assigned a unique string as part of its name so that files can't be overwritten.
3. To use this data in subsequent compiles:
   ```
   opencc -O3 -ipa -fb-opt mydata -o myexe mycode.c
   ```

The new executable doesn't contain any instrumentation library calls so it should run faster than a non-FDO '`myexe`'. You may need to experiment to see if FDO can significantly improve the performance of your application.

5.6 Aggressive Optimizations

The x86 Open64 compiler has a range of optimizations. Optimizations that can produce output that is identical to the original are classified as "safe." Optimizations that change the program's behavior even slightly are classified as "unsafe". The '`-O1, -O2, -O3`' flags

only perform safe optimizations. The use of unsafe optimizations often can produce a significant speed up in a program and still generate accurate results. Depending on the coding practices used, some unsafe optimizations may be safe. It is recommended that you first try safe flags in your program. As you experiment with unsafe flags, check for incorrect results and weigh the benefits of unsafe optimizations.

5.6.1 Alias Analysis

It is possible in C or Fortran programs that two variables might occupy the same memory. For example in C, two pointers may point to the same location, such that writing through one pointer changes the value of the variable pointed to by another. Although the C standard prohibits certain kinds of aliasing, many programs violate these rules. The aliasing behavior of the compiler is controlled by the '`-OPT:alias`' flag.

Aliases are hidden definitions and uses of data owing to:

- Accesses through pointer
- Partial overlapping in storage locations (such as unions in C)
- Procedure calls for non-local objects
- Raising of exceptions

The compiler usually assumes that aliasing will occur. The compiler performs alias analysis to identify when there is no aliasing, so subsequent optimizations can be done. Certain C/C++ rules allow for some level of alias analysis. Fortran has additional rules that eliminate aliasing in more situations. For example, subroutine parameters have no alias and side effects of calls are limited to global variables and actual parameters.

For C or C++, making certain changes in your code can assist the compiler in making the correct assumptions. Changes such as using type qualifiers like `const, restrict`, or `volatile`. If you instruct the compiler about which assumptions to make about your program, then more optimizations can be applied. The following options are listed in order of increasingly severe, and potentially dangerous, assumptions you can tell the compiler to make about your program. (Place `no_` before any value for the opposite assertion. For example, '`-OPT:alias=no_restrict`'.)

'`-OPT:alias=any`'
: Implies that any two memory references can be aliased. This is the default level.

'`-OPT:alias=typed`'
: Activates the ANSI rule that if objects have different base types, then they are not aliased. The '`-Ofast`' flag actives this option.

'`-OPT:alias=unnamed`'
: Assumes that pointers never point to named objects.

'`-OPT:alias=restrict`'
: Instructs the compiler to assume that all pointers are restricted pointers and point to distinct non-overlapping objects. The compiler is now able to invoke as many optimizations as if the program were written in Fortran. A restricted pointer behaves as if the C `restrict` keyword was used with it.

'`-OPT:alias=disjoint`'
: Instructs the compiler that any two pointer expressions are assumed to point to distinct, non-overlapping objects.

'`-OPT:alias=cray_pointer`'
: Means that an object pointed to by a Cray pointer is never overlaid on another variable's storage. Also, this flag instructs the compiler to assume that the pointed-to-object is stored in memory before a call to an external procedure, is read out of memory at its next reference, and is stored before an `END` or `RETURN` statement of a subprogram.

'`-OPT:alias=parm`'
: Instructs the compiler that Fortran parameters do not alias to any other variable. This is the default.

5.6.2 Numerically Unsafe Optimizations

Your results can be slightly changed if you rearrange mathematical expressions and change the order or number of floating point operations. For example:

```
A = 3. * X
B = 6. * Y
C = 3. * (X + 2. * Y)
```

A good compiler recognizes that `C = A + B`. However, because the order of operations is different, the results will be a slightly different C. The '`-OPT:roundoff`' flag controls this type of transformation, but there are other unsafe flags such as the following:

'`-OPT:roundoff=N, -OPT:IEEE_arithmetic=N`'
: These options control IEEE behavior.

'`-OPT:div_split=(ON | OFF)`'
: Controls transforming expressions of the form X/Y into X* (1/Y). The reciprocal is faster, but is inherently less accurate than a straight division.

'`-OPT:recip=(ON | OFF)`'
: Allows expressions of the form 1/X to be changed to use the reciprocal instruction of the computer. This may be faster, but is inherently less accurate than a division.

5.6.3 Fast-math Functions

If you want the compiler to use fast versions of math functions tuned for the processor, specify '`-OPT:fast_math=on`'. Some of the affected math functions are: `log, exp, sin, cos, sincos, expf, and pow`. In general, the accuracy is within 1 ulp of the fully precise result, but the accuracy in some cases may be worse. The routines may not set IEEE exception flags and call error handlers. The denormal number inputs/outputs are usually treated as 0, but may produce unexpected results. When '`-OPT:roundoff`' is set to 2 or above, '`-OPT:fast_math=on`' is effected. Other fast-math options that apply to all languages are:

'`-ffast-math`'
: Improves FP speed by relaxing ANSI and IEEE rules. This flag implies '`-OPT:IEEE_arithmetic=2 -fno-math-errno.`'

'`-fno-fast-math`'

Instructs the compiler to conform to ANSI and IEEE math rules even if it means a slower speed. This flag implies '`-OPT:IEEE_arithmetic=1 -fmath-errno.`'

Both '`-OPT:fast_math=on`' and '`-ffast-math`' are implied by '`-Ofast`'.

5.6.4 IEEE 754 Compliance

You can control the level of IEEE 754 compliance through options. By relaxing the level of compliance, the compiler is given greater latitude to transform the code for improved performance.

5.6.4.1 Arithmetic

In some cases, you can allow the compiler to use operations that deviate from the IEEE 754 standard to achieve significantly improved performance, while still getting results that meet the accuracy requirements of your application. To regulate the level of conformance to ANSI/IEEE 754-1985 floating-point roundoff and overflow behavior, use the following flag:

'`-OPT:IEEE_arithmetic=`*N*'

where N is:

1 Requires strict conformance to the standard.

2 Allows the use of any operation as long as exact results are produced. This is the default level at '`-O3`'. Setting N=2 allows for less accurate inexact results. For example, X*0 may be replaced by 0 and X/X may be replaced by 1 even though these replacements are inaccurate when X is +inf, -inf, or NaN.

3 Allows the use of any mathematically valid transformations. For example, x/y can be replaced by x*(recip (y)).

5.6.4.2 Roundoff

The following flag is used to identify the extent of the roundoff error that the compiler is allowed to introduce:

'`-OPT:roundoff=N`' where N is:

0 No roundoff error

1 Limited roundoff error allowed

2 Allow roundoff error caused by re-associating expressions

3 Allow any roundoff errors

The default roundoff level is 0 with optimization levels '`-O0, -O1`', and '`-O2`'. The default roundoff level is 1 with optimization level '`-O3`'.

To better understand what these levels mean, the other '`-OPT:`' sub-options that are activated by various roundoff levels are listed below:

'`-OPT:roundoff=1`' implies:

- '`-OPT:fast_exp=ON`' enables optimization by replacing the runtime call for exponentiation by multiplication and/or square root operations for certain compile-time constant exponents (integers and halves).
- '`-OPT:fast_trunc`' implies inlining of Fortran intrinsics: `NINT, ANINT, AINT`, and `AMOD`.

'`-OPT:roundoff=2`' activates the following suboption:

- '`-OPT:fold_reassociate`' allows optimizations involving re-association of floating-point quantities.

'`-OPT:roundoff=3`' activates the following suboptions:

- '`-OPT:fast_complex`'. When this option is activated, complex absolute value (norm) and complex division use fast algorithms that overflow for an operand (the divisor, in the case of division) that has an absolute value that is larger than the square root of the largest representable floating-point number.
- '`-OPT:fast_nint`' uses a hardware feature to implement single- and double-precision versions of `NINT` and `ANINT`.

5.6.5 Additional Unsafe Optimizations

There are some advanced optimizations that exploit certain instructions, such as CMOVE (conditional move), which result in slightly changed program behavior. For example, programs that write into variables guarded by an if () statement:

```
if (a == 1) then
    a=7
endif
```

Here, the fastest code on an x86 processor is one that avoids a branch by always writing a; if the condition is false, it writes a's existing value into a, else it writes 7 into a. If a is a read-only value not equal to 1, this optimization causes a segmentation fault in this strange, but valid program.

5.6.6 Numerical Accuracy Assumptions

The following table lists the assumptions made about numerical accuracy at different levels of optimization.

-OPT: option name	-O0	-O1	-O2	-O3	-Ofast	Notes
div_split	off	off	off	off	on	on if: IEEE_arith=3
fast_complex	off	off	off	off	off	on if: roundoff=3
fast_exp	off	off	off	on	on	on if: roundoff>=1
fast_nint	off	off	off	off	off	on if: roundoff=3
fast_sqrt	off	off	off	off	off	-
fast_trunc	off	off	off	on	on	on if: roundoff>=1
fold_reassociate	off	off	off	off	on	on if: roundoff>=2
fold_unsafe_relops	off	off	off	on	on	-
fold_unsigned_relops	off	off	off	off	off	-
IEEE_arithmetic	1	1	1	2	2	-
IEEE_NaN_inf	off	off	off	off	off	-
recip	off	off	off	off	on	on if: roundoff>=2
roundoff	0	0	0	1	2	-
fast_math	off	off	off	off	off	on if: roundoff>=2
rsqrt	0	0	0	0	1	1 if: roundoff>=2

Table C: Numerical Accuracy with Options

For example, if you use '`-OPT:roundoff`' at '`-O3`', the flag is set to '`-OPT:roundoff=2`' by default.

5.6.6.1 Flush-to-Zero Behavior

To increase the speed of the processor hardware that implements IEEE floating-point arithmetic, the compiler allows it to generate zero rather than a denormalized number when an arithmetic operation underflows. At optimization level '`-O3`', the compiler allows this behavior, which is known as *flush to zero*. The flag that controls flush-to-zero behavior is:

'`-OPT:IEEE_arith=N`'
N=2 or 3 - Allows flush-to-zero behavior
This flag defaults to 1 under optimization levels -O0, -O1, and -O2. It defaults to 2 under optimization level -O3. The compilation flag works by generating instructions to perform the setting at the entry to `main()`.

During runtime, you can set it by using the `IEEE_SET_UNDERFLOW_MODE` Fortran intrinsic found in the intrinsic module `IEEE_ARITHMETIC`:

```
! Gradual underflow means "produce denormalized numbers"
USE, INTRINSIC :: IEEE_ARITHMETIC
CALL IEEE_SET_UNDERFLOW_MODE(GRADUAL=.TRUE.)
```

5.7 Hardware Performance

There are a number of ways to configure your hardware and software that can cause substantial performance degradations. The following sections describe techniques for getting the best performance from your system.

5.7.1 Memory Setup

The number, type, and placement of memory modules on a motherboard can each affect the memory latency and bandwidth that can be achieved. Refer to your motherboard manual for information about the effects of memory placement in different slots. Incorrectly setting up your memory can account for up to a factor-of-two difference in memory performance and can even affect system stability.

5.7.2 BIOS Setup

Your system BIOS may allow you to change your motherboard's memory interleaving options. This may have an effect on your system performance depending on your configuration.

5.7.3 Multiprocessor Memory

Traditional small multiprocessor systems use symmetric multiprocessing (SMP), where memory latency and bandwidth is the same for all processors. This is not the case for AMD Opteron^{TM} multiprocessor systems which provide non-uniform memory access (NUMA). On these systems, each processor has its own direct-attached memory. Although each processor can access the memory of all others, memory that is physically closest has both the lowest latency and highest bandwidth. The larger the number of processors, the higher the latency and the lower the bandwidth between the two processors that are physically furthest apart.

The BIOS for most multiprocessors allow you to enable or disable interleaving of memory across nodes. It is recommended that you disable node interleaving because while memory interleaving across nodes masks the NUMA variation in behavior, it results in uniformly lower performance.

5.7.4 Kernel and System Effects

For the best performance on a NUMA system, a process (thread) and as much memory that it uses must be allocated to the same processor. Historically, the Linux kernel had no support for setting the affinity of a process in this way.

Running a non-NUMA kernel on a NUMA system can result in changes in performance while a program is running, because a kernel will schedule a process to run on whatever processor is free without considering where the process memory is allocated. Recent kernels do support NUMA to some degree. These kernels will attempt to allocate memory local to the processor where a thread is running. However later on, they may run that same thread on a different processor after memory has been allocated. Current NUMA-supported kernels do not migrate memory across NUMA nodes. This means that if a process moves relative to its memory, its performance will degrade in unpredictable ways. Check with your distribution vendor to find out if your kernel supports NUMA and you have C libraries that can interface with them.

5.7.5 Tools and APIs

To workaround the problem of the kernel moving a process away from its memory, recent Linux distributions include tools and APIs that allow you to bind a thread to run on a specific processor. If your Linux distribution comes with a package called '`schedutils`', there is a program called "taskset". With taskset, you can specify that a program must run on one particular processor.

For low-level programming, the `sched_setaffinity (2)` call in the C library can provide this capability. To use this call, you'll need a recent C library. On systems where the kernel doesn't support NUMA and on runs that do not set process affinity before they start, performance can vary by 30% or more between individual runs.

5.7.6 Testing Memory Latency and Bandwidth

To test your memory latency and bandwidth, try the following tools:

- For memory latency, LMbench package provides a tool called lat_mem_rd. This tool provides a view of your memory hierarchy latency. It is available from: `http://www.bitmover.com/lmbench/`
- For measuring memory bandwidth, use the STREAM benchmark tool. It is available from: `http://www.streambench.org/`

  Use the following command lines when compiling either the Fortran or C version of the benchmark:

  ```
  $ openf95 -Ofast stream.f mysecond.c
  $ opencc -Ofast stream.c
  ```

  (If you do not compile with at least '`-O3`', performance may drastically drop.)

For both tools, you should perform a number of identical runs and average your results. There may be variations of more than 10% between runs.

5.8 Displaying How the Compiler Optimized Code

If you are curious how the compiler optimized your code, there are several ways to generate a listing showing (by line number) what optimizations the compiler performed.

5.8.1 Using the '-S' Flag

Use the '-S' flag to see what the compiler did. This way is especially useful if you understand some assembly code. The following example uses the STREAM benchmark. First, compile STREAM using this command line:

```
$opencc -Ofast stream.c -S
```

This produces a 'stream_d.s' assembly file. In this file you can see sections of readable comments interspersed with assembly code that looks like the following:

```
.LBB6_tuned_STREAM_Scale:
 #<loop> Loop body line 410, nesting depth: 1, iterations: 250000
 #<loop> unrolled 4 times
.loc 1 414 0
 # 411   int j;
 # 412  #pragma omp parallel for
 # 413   for (j=0; j<N; j++)
 # 414       b[j] = scalar*c[j];
movdqa %xmm1,%xmm4              # [0]
mulpd 0(%rax),%xmm4             # [2]
movdqa %xmm1,%xmm3              # [7]
movntpd %xmm4,0(%rsi)           # [9] id:8 b+0x0
mulpd 16(%rax),%xmm3            # [9]
movdqa %xmm1,%xmm2              # [14]
movntpd %xmm3,16(%rsi)          # [16] id:8 b+0x0
mulpd 32(%rax),%xmm2            # [16]
movdqa %xmm1,%xmm0              # [21]
movntpd %xmm2,32(%rsi)          # [23] id:8 b+0x0
mulpd 48(%rax),%xmm0            # [23]
addq $64,%rax                   # [28]
addq $64,%rsi                   # [29]
cmpq %rdx,%rax                  # [29]
prefetchnta 576(%rax)           # [30] L1
movntpd %xmm0,-16(%rsi)         # [30] id:8 b+0x0
jle .LBB6_tuned_STREAM_Scale    # [30]
```

Note the "unrolled 4 times" comment and the original source in comments, which describes what the compiler did (without having to understand the x86 assembly code).

5.8.2 -CLIST or -FLIST

To see what the compiler is doing, you can use:
'-CLIST:=on' for C codes
'-FLIST:=on' for Fortran code

On the STREAM source code used in the previous example, compile using this command line:

```
$ opencc -O3 CLIST:=ON -c stream.c
```

The output looks like:

```
/opt/open64/lib/gcc-lib/x86_64-open64-linux/4.2/be translates /tmp/ccI.16xQZJ into
  stream.w2c.h and stream.w2c.c, based on source stream.c
```

If you look at 'stream.w2c.c' with an editor, you will see some strange looking C code. In this example, there wasn't much optimizing being done, but in codes where LNO (Loop Nest Optimization) is important, you would see a lot more optimizations.

5.8.3 Verbose Flags

To see vectorization activity, you can also turn on the verbose flags in LNO. Use the '`-LNO:simd_verbose`' flag in the command line:

```
$ opencc -O3 -LNO:simd_verbose -c stream.c
```

The output looks like:

```
(stream_d.c:103) LOOP WAS VECTORIZED.
(stream_d.c:119) LOOP WAS VECTORIZED.
(stream_d.c:142) LOOP WAS VECTORIZED.
(stream_d.c:147) LOOP WAS VECTORIZED.
(stream_d.c:152) LOOP WAS VECTORIZED.
(stream_d.c:157) LOOP WAS VECTORIZED.
(stream_d.c:164) Nonvectorizable ops/non-unit stride.
   Loop was not vectorized.
(stream_d.c:211) Nonvectorizable ops/non-unit stride.
   Loop was not vectorized.
```

The above example tells you more about what the compiler is doing with loops. Also, you can use the '`-LNO:vintr_verbose`' flag in the command line:

```
$opencc -O3 -LNO:vintr_verbose -c stream.c
```

In this example, the output doesn't tell you much because there are no intrinsic functions to get vectorized in STREAM.

5.9 OpenMP and Autoparallelization

The user can easily create programs for shared memory computers from new or existing code by using the compiler directives defined by the OpenMP Application Program Interface (API). OpenMP provides parallel directives and library routines that comply with the OpenMP API specification 2.5. The x86 Open64 compiler includes OpenMP and autoparallelization for both Fortran and C/C++.

5.9.1 OpenMP

OpenMP provides a portable/scalable interface that is in effect the standard for programming shared memory computers (processors that share physical memory). OpenMP enables the user to create threads, assign work to threads, and manage data within a program. The compiler enables incremental parallelization of your code on shared memory processor (SMP) systems. This means you can add directives to parts of existing code a little at a time. Directives enable you to distribute the work of your application over several processors. OpenMP supports the following features of parallel programming:

- Specification of parallel execution
- Communication between multiple threads
- Synchronization between threads

The optimal number of threads to be executed in parallel for the platform's multiple processors is automatically created by the OpenMP library. The number of threads is typically equal to the number of processors in the system. For systems with only one processor, you will not see a speedup. Your program may even run slower due to the overhead in the synchronization code generated by the compiler.

Under parallel execution, the speedup depends mostly on the algorithms you use and the way you use the OpenMP directives. If your program exhibits a high degree of coarse grain

parallelism, the speedup will be significant as the number of processors are increased. For more information on OpenMP, go to the OpenMP home page: `http://openmp.org/wp/`

5.9.2 Autoparallelization

When autoparallelization is used, the compiler attempts to parallelize program code without depending on user directives. You invoke autoparallization by specifying the '`-apo`' option in the compile and link lines as shown in the following example:

```
$ openf95 ... -apo .... -c foo.f95
$ openf95 ... -apo .... -o foodata foo.o data.o
```

OpenMP directives are helpful because the compiler is only able to parallelize a subset of the loops that you know are parallelizable. You must specify '`-mp`' for the compiler to recognize OpenMP directives. For programs containing OpenMP directives, you can combine autoparallelization with OpenMP directives to additionally parallelize code that does not contain OpenMP directives. For these cases, you should specify the '`-apo`' and '`-mp`' options together as shown in the following example:

```
$ openf95 ... -apo -mp .... -c foo.f95
$ openf95 ... -apo -mp .... -o foodata foo.o data.o
```

Currently, the compiler only uses OpenMP directives for autoparallelization analysis. The extent of the benefit of autoparallelization varies with the program's characteristics and the data set used. In a few cases, autoparallelization can cause a slight performance degradation of the program because an autoparallelized program runs under multiple threads. The overhead during execution is due to the runtime decision to create multiple threads, followed by their synchronization.

To facilitate its decision making, the compiler generates both a serial and a parallel version when parallelizing a loop. At runtime, the generated code decides whether to execute the serial or the parallel version by looking at the total amount of work performed by the loop. The serial version will be executed if the amount of work is not large enough to justify the additional synchronization overhead. In these cases, due to the need to make the decision at run time, the performance is slower than if the program is not compiled with '`-apo`'.

You can use the '`-LNO:parallel_overhead`' option to control the synchronization overhead. The value is the compiler's estimate of the overhead in processor cycles in invoking the parallel version of a loop. This value affects the runtime decision on which version to use because the optimal value varies by system and program. You can also use this option for parallel performance tuning under '`-apo`'.

5.9.3 Starting OpenMP

The OpenMP process is as follows:

1. Add directives where appropriate.
2. Compile and link your code using the '`-mp`' flag. This flag instructs the compiler to recognize the OpenMP directives in your program and processes the source code guarded by the OpenMP conditional compilation sentinels (e.g., #pragma for C/C++ code and !$ for Fortran). The setting of the OpenMP Environment Variables also affects the actual program execution.
3. The compiler generates different output that results in the program running in multiple threads during execution. The output code is linked with the OpenMP Runtime Library for execution under multiple threads.

Mistakes can be made in the inserted OpenMP code resulting in incorrect execution. This happens because the OpenMP directives tell the compiler what constructs in the program can be parallelized and how to parallelize them. To check for these mistakes, first ensure that all OpenMP-related code is guarded by conditional compilation sentinels, then recompile the same program without the '`-mp`' flag. The resulting executable will be run serially. If your program runs with no errors, the problems in parallel execution are due to mistakes in the OpenMP part of the code. This should make tracking down and fixing the problem easier.

5.9.4 OpenMP Directives for Fortran

Start all OpenMP directives for Fortran with comment characters followed by `$OMP` or `$omp`. The compiler only processes these directives if '`-mp`' is specified. Possible comment characters include: !, C, c, and *. For fixed-form Fortran, the `!$OMP` directives must begin in the first column of the line.

Following is a list of Fortran compiler directives provided by version 2.5 of the OpenMP Fortran API.

Parallel region construct - Defines a parallel region.

`PARALLEL` Clauses:
`FIRSTPRIVATE`
`PRIVATE`
`SHARED`
`DEFAULT (PRIVATE/ SHARED/ NONE)`
`REDUCTION`
`COPYIN`
`IF`
`NUM_THREADS`

Example:

```
!$OMP parallel [clause] ...
structured-block
!$OMP end parallel
```

Work sharing constructs - Divide the execution of the enclosed block of code between the team members encountering it.

`DO` Clauses:
`(NOWAIT)`
`PRIVATE`
`FIRSTPRIVATE`
`LASTPRIVATE`
`REDUCTION`
`SCHEDULE` (static, dynamic, guided, runtime)
`ORDERED`

Example:

```
!$OMP do [clause] ...
do-loop
!$OMP enddo [nowait]
```

`SECTIONS` Clauses:
`PRIVATE`
`FIRSTPRIVATE`
`LASTPRIVATE`
`REDUCTION`

Example:

```
!$OMP sections [clause] ...
structured-block
!$OMP end sections [nowait]
```

`SINGLE` Clauses:
`PRIVATE`
`FIRSTPRIVATE`
End Clauses:
`COPYPRIVATE`
`NOWAIT`

Example:

```
!$OMP single [clause] ...
structured-block
!$OMP end single [end clause]
```

Combined parallel work sharing constructs - Shortcut for specifying a parallel region containing only one work-sharing construct.

`PARALLEL DO`
Clauses: None
Example:

```
!$OMP parallel do
structured-block
!$OMP end parallel do
```

`PARALLEL SECTIONS`
Clauses: None
Example:

```
!$OMP parallel sections
structured-block
!$OMP end parallel sections
```

`PARALLEL WORKSHARE`
Clauses: None
Example:

```
!$OMP parallel workshare
structured-block
!$OMP end parallel workshare
```

Synchronization constructs - Enable various features of synchronization. For example, access to a block of code or execution order of statements within a block of code.

`ATOMIC` Clauses: None
Example:

```
!$OMP atomic
expression-statement
```

`BARRIER` Clauses: None
Example:

```
!$OMP barrier
```

`CRITICAL` Clauses: None
Example:

```
!$OMP critical [ (name)]
structured-block
!$OMP end critical [ (name)]
```

`FLUSH` Clauses: None
Example:

```
!$OMP flush [ (list)]
```

`MASTER` Clauses: None
Example:

```
!$OMP master
structured-block
!$OMP end master
```

`ORDERED` Clauses: None
Example:

```
!$OMP ordered
structured-block
!$OMP end ordered
```

Data environments - Control the data environment while the parallel constructs are executed.

`THREADPRIVATE`
Clauses: None
Example:

```
!$OMP threadprivate (list)
```

`WORKSHARE`
Clauses: None
Example:

```
!$OMP workshare
structured-block
!$OMP end workshare [nowait]
```

5.9.5 OpenMP Compiler Directives for C/C++

All OpenMP directives for C/C++ start with `#pragma`.

These directives are only processed by the compiler if '`-mp`' is specified. Following is a list of C and C++ compiler directives provided by version 2.5 of the OpenMP C/C++ API.

Parallel region construct - Defines a parallel region.

`PARALLEL` Clauses:
`PRIVATE`
`SHARED`
`FIRSTPRIVATE`
`DEFAULT (SHARED/NONE)`
`REDUCTION`

COPYIN
IF
NUM_THREADS

Example:

```
#pragma omp parallel [clause] ...
structured-block
```

Work sharing constructs - Divide the execution of the enclosed block of code between team members encountering it.

FOR

Clauses:
NOWAIT
PRIVATE
FIRSTPRIVATE
LASTPRIVATE
REDUCTION
SCHEDULE (static, dynamic, guided, runtime)
ORDERED

Example:

```
#pragma omp for [clause] ...
for-loop
```

SECTIONS

Clauses:
NOWAIT
PRIVATE
FIRSTPRIVATE
LASTPRIVATE
REDUCTION

Example:

```
#pragma omp sections [clause] ...
structured-block
```

SINGLE

Clauses:
NOWAIT
PRIVATE
FIRSTPRIVATE
COPYPRIVATE

Example:

```
#pragma omp single [clause] ...
structured-block
```

Combined parallel work sharing constructs - Shortcut for specifying a parallel region containing only one work-sharing construct.

`PARALLEL FOR`
Clauses: None
Example:

```
#pragma omp parallel for
structured-block
```

`PARALLEL SECTIONS`
Clauses: None
Example:

```
#pragma omp parallel sections
structured-block
```

Synchronization constructs - Enable various features of synchronization. For example, access to a block of code or execution order of statements within a block of code.

`ATOMIC` Clauses: None
Example:

```
#pragma omp atomic
expression-statement
```

`BARRIER` Clauses: None
Example:

```
#pragma omp barrier
```

`CRITICAL` Clauses: None
Example:

```
#pragma omp critical [ (name)]
structured-block
```

`FLUSH` Clauses: None
Example:

```
#pragma omp flush [ (list)]
```

`MASTER` Clauses: None
Example:

```
#pragma omp master
structured-block
```

`ORDERED` Clauses: None
Example:

```
#pragma omp ordered
structured-block
```

Data environments - Control the data environment while the parallel constructs are executed.

`THREADPRIVATE`
Clauses: None
Example:

```
#pragma omp threadprivate(list)
```

5.9.6 OpenMP Runtime Library Calls for Fortran

Standard routines implemented in the OpenMP runtime library can be explicitly called by OpenMP programs. To ensure your program is still compilable without '`-mp`', guard your code with the OpenMP conditional compilation sentinels.

Following is a list of OpenMP runtime library routines provided by version 2.5 of the OpenMP Fortran API.

`call omp_set_num_threads (integer)`
: Sets the number of threads to use in a team.

`integer omp_get_num_threads ()`
: Returns the number of threads in the currently executing parallel region.

`integer omp_get_max_thread ()`
: Returns the maximum value that `omp_get_num_threads` may return.

`integer omp_get_thread_num ()`
: Returns the thread number within the team.

`integer omp_get_num_procs ()`
: Returns the number of processors available to the program.

`call omp_set_dynamic (logical))`
: Controls the dynamic adjustment of the number of parallel threads.

`logical omp_get_dynamic ()`
: If dynamic threads are enabled, returns .TRUE.; otherwise returns .FALSE.

`logical omp_in_parallel ()`
: For calls within a parallel region, returns .TRUE.; otherwise returns .FALSE.

`call omp_set_nested (logical)`
: Enables or disables nested parallelism.

`logical omp_get_nested ()`
: If nested parallelism is enabled, returns .TRUE.; otherwise returns .FALSE.

Following is a list of lock routines:

`omp_init_lock (int)`
: Allocates and initializes lock. It is associated with the lock variable passed in as a parameter.

`omp_init_nest_lock (int)`
: Initializes a nestable lock. It is associated with a specified lock variable.

`omp_set_lock (int)`
: Acquires the lock. If necessary, it will wait until it becomes available.

`omp_set_nest_lock (int)`
: Sets a nestable lock. The thread executing the subroutine waits until a lock becomes available. It then sets that lock and increments the nesting count.

`omp_unset_lock (int)`
: Releases the lock. It resumes a waiting thread if there are any.

`omp_unset_nest_lock (int)`
: Releases ownership of a nestable lock. The subroutine decrements the nesting count and then releases the associated thread from the ownership of the nestable lock.

`logical omp_test_lock (int)`
: Attempts to acquire the lock. If successful, returns .TRUE.; otherwise returns .FALSE.

`omp_test_nest_lock (int)`
: Tries to set a lock using the same method as `omp_set_nest_lock` but the execution thread doesn't wait for confirmation that the lock is available. If the lock is successfully set, function increments the nesting count. If the lock is unavailable, function returns a value of zero.

`omp_get_wtime`
: Returns double-precision value equal to the number of seconds since the initial value of the OS real-time clock.

`omp_get_wtick`
: Returns double-precision floating-point value equal to the number of seconds between successive clock ticks.

5.9.7 OpenMP Runtime Library Calls for C/C++

Standard routines implemented in the OpenMP runtime library can be explicitly called by OpenMP programs. To ensure your program is still compilable without '`-mp`', guard your code with the OpenMP conditional compilation sentinels (e.g., `#pragma`).

Following is a list of OpenMP runtime library routines provided by version 2.5 of the OpenMP C/C++ API.

`void omp_set_num_threads (int)`
: Sets the number of threads to use in a team.

`int omp_get_num_threads (void)`
: Returns the number of threads in the currently executing parallel region.

`int omp_get_max_thread (void)`
: Returns the maximum value that `omp_get_num_threads` may return.

`int omp_get_thread_num (void)`
: Returns the thread number within the team.

`int omp_get_num_procs (void)`
: Returns the number of processors available to the program.

`void omp_set_dynamic (int)`
: Controls the dynamic adjustment of the number of parallel threads.

`int omp_get_dynamic (void)`
: If dynamic threads are enabled, returns a non-zero value; otherwise returns 0.

`int omp_in_parallel (void)`
: For calls within a parallel region, returns a non-zero value; otherwise returns 0.

`void omp_set-nested (int)`
Enables or disables nested parallelism.

`int omp_get_nested (void)`
If nested parallelism is enabled, returns a non-zero value; otherwise returns 0.

Following is a list of lock routines:

`omp_init_lock (omp_lock_t *)`
Allocates and initializes lock, associating it with the lock variable passed in as a parameter.

`omp_init_nest_lock (omp_nest_lock_t *)`
Initializes a nestable lock and associates it with a specified lock variable.

`omp_set_lock (omp_lock_t *)`
Acquires the lock. If necessary, waits until it becomes available.

`omp_set_nest_lock (omp_nest_lock_t *)`
Sets a nestable lock. The thread executing the subroutine waits until a lock becomes available. It then sets that lock and increments the nesting count.

`omp_unset_lock (omp_lock_t *)`
Releases the lock. It resumes any waiting thread.

`omp_unset_nest_lock (omp_nest_lock_t *)`
Releases ownership of a nestable lock. The subroutine first decrements the nesting count. It then releases the associated thread from the ownership of the nestable lock.

`int omp_test_lock (omp_lock_t *)`
Attempts to acquire the lock. If successful, returns a non-zero value; otherwise returns 0.

`omp_test_nest_lock (omp_nest_lock_t *)`
Tries to set a lock using the same method as `omp_set_nest_lock` but the execution thread doesn't wait for confirmation that the lock is available. If setting the lock is successful, function increments the nesting count. If lock is unavailable, function returns 0.

`double omp_get_wtime (void)`
Returns double-precision value equal to the number of seconds since the initial value of the OS real-time clock.

`double omp_get_wtick (void)`
Returns double-precision floating-point value equal to the number of seconds between successive clock ticks.

5.9.8 Runtime Libraries

For each library, there are both static and dynamic versions and both 64-bit and 32-bit versions. The libraries are:

– Dynamic 64-bit:
'`<installdir>/lib/gcc-lib/x86_64-open64-linux/<version>/libopenmp.so`'

– Static 64-bit:
'<installdir>/lib/gcc-lib/x86_64-open64-linux/<version>/libopenmp.a'
– Dynamic 32-bit:
'<installdir>/lib/gcc-lib/x86_64-open64-linux/<version>/32/libopenmp.so'
– Static 32-bit:
'<installdir>/lib/gcc-lib/x86_64-open64-linux/<version>/32/libopenmp.a'

The symbolic links to the dynamic versions of the libraries for both 32- and 64-bit environments are:

– Symbolic link to 64-bit dynamic version:
'<idir>/lib/gcc-lib/x86_64-open64-linux/<version>/libopenmp.so.1'
– Symbolic link to 32-bit dynamic version:
'<idir>/lib/gcc-lib/x86_64-open64-linux/<version>/32/libopenmp.so.1'

Remember to use the '-mp' flag on both the compile and link lines.

5.9.9 Environment Variables

You can change the execution behavior of a program running under multiple threads by using OpenMP environment variables. You set these variables by using the shell commands, for example:

bash:

```
export OMP_NUM_THREADS=4
```

In csh:

```
setenv OMP_NUM_THREADS 4
```

After the above commands, this command will display the value 4:

```
echo $OMP_NUM_THREADS 4
```

The available environment variables to use with OpenMP are:

`OMP_DYNAMIC`
: Specify whether to enable or disable dynamic adjustment of the number of threads available.

`OMP_NESTED`
: Specify whether to enable or disable nested parallelism.

`OMP_SCHEDULE`
: Specify the schedule type to be applied to `DO` and `PARALLEL_DO` directives with RUNTIME schedule type. `OMP_SCHEDULE` can be any of STATIC, DYNAMIC, or GUIDED.

`OMP_NUM_THREADS`
: Specify the number of threads to be used during execution.

`OMP_SLAVE_STACK_SIZE`
: Specify the amount of stack size to be used for slave threads.

`O64_OMP_SET_AFFINITY`
: Specify if the operating system's affinity mechanism is used to assign OpenMP threads to CPUs. Assignment of threads to processors occurs if the variable is set to `TRUE` or if the variable is not set. Assignment of threads to processors is disabled if the variable is set to `FALSE`.

`O64_OMP_SPIN_COUNT`
: Specify the number of times the spin loops will spin at user-level before falling back to operating system schedule/reschedule mechanisms. The default value is 20000.

`O64_OMP_SPIN_USER_LOCK`
: Specify whether or not to use user-level spin mechanism for OpenMP locks. If the variable is set to `TRUE` then user-level spin mechanisms are used. If the variable is set to `FALSE` then pthread mutexes are used. The default if the variable is not set is the same as `FALSE`.

`O64_OMP_AFFINITY_MAP`
: Specify the thread-CPU relationship when the operating system's affinity mechanism is used to assign OpenMP threads to CPUs. The variable should contain a list of CPUs. For example, `O64_OMP_AFFINITY_MAP="3 1 2 0"` maps thread 0 to CPU 3, thread 1 to CPU 1, thread 2 to CPU 2 and thread 3 to CPU 0.

5.9.10 C/C++ Example Using OpenMP Directives

The following parallel version of the "hello world" program uses OpenMP directives. It spawns multiple threads when run. To ensure that printing from the various threads do not overwrite one another, the program uses the `CRITICAL` directive.

```
#include <omp.h>
main()
{
   int tid = 0;
   int nthreads = 1;
/* Fork a team of threads giving them their own copies of variable tid */

   #pragma omp parallel private (tid)
  {
#ifdef _OPENMP
   /* Obtain and print thread id */
   tid = omp_get_thread_num ();
#endif
   #pragma omp critical
     printf ("Hello World from thread %d\n", tid);
   #pragma omp master
   #pragma omp critical
     {
#ifdef _OPENMP
   /* Only master thread does this */
     nthreads = omp_get_num_threads ();
#endif
     printf ("Number of threads = %d\n", nthreads);
     }
   /* All threads join master thread and disband */
   }
}
```

Some lines begin with `#pragma` and `#ifdef` which are the conditional compilation tokens. Without '`-mp`', these lines are ignored when compiled.

Use the following command to compile '`omphello.c`' for OpenMP:

```
$ opencc -c -mp omphello.c
```

Use the following command to link it:

```
$ opencc -mp omphello.o -o  omphello.out
```

Use the following command to set the environment variable for the number of threads:

```
$ export OMP_NUM_THREADS=5
```

The output after running the program is:

```
$ ./omphello.out
  Hello World from thread1
  Hello World from thread2
  Hello World from thread3
  Hello World from thread0
  Number of threads = 5
  Hello World from thread4
```

Each time you run the program, the output from the different threads can be in a different order. To run with only two threads, change the environment variable to:

```
$ export OMP_NUM_THREADS=2
```

Now the output is:

```
$ ./omphello.out
  Hello World from thread0
  Number of threads = 2
  Hello World from thread1
```

You can compile and link the same program without '`-mp`'. In doing so, the directives will be ignored. To compile without '`-mp`', use:

```
$ opencc -c omphello.c
```

Use the following command to link it:

```
$ opencc omphello.o -o  omphello.out
```

Now the output is:

```
$ ./omphello.out
  Hello World from thread0
  Number of threads = 1
```

5.9.11 Fortran Example Using OpenMP Directives

The following parallel version of the "hello world" program uses OpenMP directives. It spawns multiple threads when run. To ensure that printing from the various threads do not overwrite one another, the program uses the `CRITICAL` directive.

```
      PROGRAM HELLO
      INTEGER NTHREADS, TID, OMP_GET_NUM_THREADS, OMP_GET_THREAD_NUM
      TID=0
      NTHREADS=1
!     Fork a team of threads giving them their own copies
!     of variables TID PARALLEL PRIVATE (TID)
!     Obtain and print thread id
!$    TID = OMP_GET_THREAD_NUM()
!$OMP CRITICAL
      PRINT *, 'Hello World from thread ', TID
!$OMP END CRITICAL
```

```
!$OMP MASTER
!$OMP CRITICAL
!     Only master thread does this
!$    NTHREADS= OMP_GET_NUM_THREADS ()
      PRINT *, 'Number of threads = ', NTHREADS
!$OMP END CRITICAL
!$OMP END MASTER
!     All threads join master thread and disband
      END
```

Some lines begin with `!$` which are the conditional compilation tokens. Without '`-mp`', these lines are ignored when compiled.

Use the following command to compile '`omphello.f`' for OpenMP:

```
$ openf95 -c -mp omphello.f
```

Use the following command to link it:

```
$ openf95 -mp omphello.o -o  omphello.out
```

Use the following command to set the environment variable for the number of threads:

```
$ export OMP_NUM_THREADS=5
```

The output after running the program is:

```
$ ./omphello.out
  Hello World from thread1
  Hello World from thread2
  Hello World from thread3
  Hello World from thread0
  Number of threads = 5
  Hello World from thread4
```

Each time you run the program, the output from the different threads can be in a different order. To run with only two threads, change the environment variable to:

```
$ export OMP_NUM_THREADS=2
```

Now the output is:

```
$ ./omphello.out
  Hello World from thread0
  Number of threads = 2
  Hello World from thread1
```

You can compile and link the same program without '`-mp`'. In doing so, the directives will be ignored. To compile without '`-mp`', use:

```
$ openf95 -c omphello.f
```

Use the following command to link it:

```
$ openf95 omphello.o -o  omphello.out
```

Now the output is:

```
$ ./omphello.out
  Hello World from thread0
  Number of threads = 1
```

5.9.12 Tuning

It is a good idea to first build a serial version of your application and tune the serial performance. Frequently, flags that work well for serial performance, work equally well for OpenMP performance. Usually, OpenMP will coarsely parallelize the outer iterations of the compute-intensive loops. This leaves parts of the outer and inner loops that often behave

like the serial code. Once you find good options for the serial parts of your code, you can consider OpenMP-specific issues, such as scaling, scheduling, and affinity. In some cases, you may need to tune the flags with OpenMP enabled before completing the serial version because these test cases may take a long time to run or need large amounts of memory.

5.9.12.1 Reducing the Size of Data Sets

When you want to quickly determine the efficacy of a particular tuning option, you may consider improving runtime by reducing the size of the data sets. However, because the fork/join overhead diminishes as the loops get larger, you will find that OpenMP performance will improve with larger data sets. So it is a good idea to also run tests with the full data set, especially for scaling issues. Another way to speed up is to make use of more memory and more cache on an n-way multiprocessor than a uniprocessor.

5.9.12.2 Enabling OpenMP

Use the '`-mp`' flag to turn on OpenMP parallelization after you have completed tuning the serial version of your application. To see how your application scales, try running the code on different numbers of processors. Turning on the option '`-OPT:early_mp`' is very important for OpenMP tuning. (The default setting is off.) This flag determines the order of the (SIMD) vectorization and OpenMP parallelization optimization phase of the compiler as follows:

- With early MP, first loops are parallelized, then these parallel loops are vectorized.
- With late MP, first loops are vectorized, then the vectorized loops are parallelized.

Sometimes the order will make a difference, so you need to try both.

5.9.12.3 Optimizations for OpenMP

Trying the different optimizations on your OpenMP application is the only way to determine which optimizations will make a significant difference in performance. The optimizations that are most helpful are the ones that tend to be loop nest optimization (LNO), code generation (CG), aggressive optimizations (which reduce numerical accuracy), and interprocedural analysis (IPA).

Libraries

For applications that spend a lot of time in numerical libraries, the method of optimization depends on the number of nodes. At small numbers of nodes, try optimizing and tuning a serial algorithm written for the target processor. This method may perform better than a parallel implementation based on a non-optimized algorithm. At larger numbers of nodes, a parallel version may scale and improve performance. Typically, OpenMP parallelization of the best serial algorithm (which exploit features such as SSE) provides the best performance. It is always a good idea to check if there are OpenMP-enabled versions of these numerical libraries available.

Memory System

The performance of the memory system often impacts OpenMP applications. A good place to start optimizing is to first tune the memory system with an OpenMP version of the STREAM benchmark. Ensure that the BIOS settings for memory bank interleaving is set

to AUTO, and node interleaving is set to OFF. Interleaving memory by node is not needed because OpenMP programs have good memory locality. Thus the better method is to use NUMA optimizations in the operating system to optimize the placement of data relative to threads. This optimization relies on "first touch." This means that the thread that first touches the data is assumed to be the most frequent user of this data. The memory associated with the processor that is currently running the thread is allocated this data. A NUMA-aware operating system applies this allocation at the page level. You may need a kernel upgrade to a NUMA-aware OS for good performance.

Load Balancing

The "`top`" program may give you some insight into the load balancing of your OpenMP application. You may be able to view the breakdown of user, system, and idle time per processor by pushing "1". Also try increasing the update rate (with "s" followed by 0.5). You may be able to see the program moving from serial-to-parallel phases and see if the work is being well distributed. You may discover that excessive time is spent in the system or in swapping. Always run OpenMP applications on nodes with no other applications running.

For OpenMP applications using runtime scheduling, use the `OMP_SCHEDULE` environment variable to vary the runtime schedule. Your choice of schedule and chunk size may affect performance.

Another way of gaining insight into your system performance is to use OProfile to build up a profile of the system. (The '`gprof`' profiling ('`-pg`') doesn't work with pthreads or the OpenMP library.) OProfile creates a profile that captures application code, dynamic libraries, kernel, modules, and drivers. Since OProfile can attribute the samples on a thread or CPU basis, you may discover load balancing and scheduling issues. OProfile can also access many different performance counters which can give you more insight into the behavior of the processor.

For applications using nested OpenMP parallelism, set the `OMP_NESTED` environment variable to TRUE to set the nested parallelism support.

Feedback Data

If the '`-fb-create`' option instructs an OpenMP program to generate feedback data in feedback-directed compilation, then the instrumented executable should only run under a single thread. This can be achieved by using the `OMP_NUM_THREADS` environment variable. Because the instrumentation library ('`libinstr.so`') used during execution does not support simultaneous updates of the feedback data by multiple threads, running the instrumented executable under multiple threads can cause segmentation faults.

Funding Free Software

If you want to have more free software a few years from now, it makes sense for you to help encourage people to contribute funds for its development. The most effective approach known is to encourage commercial redistributors to donate.

Users of free software systems can boost the pace of development by encouraging for-a-fee distributors to donate part of their selling price to free software developers—the Free Software Foundation, and others.

The way to convince distributors to do this is to demand it and expect it from them. So when you compare distributors, judge them partly by how much they give to free software development. Show distributors they must compete to be the one who gives the most.

To make this approach work, you must insist on numbers that you can compare, such as, "We will donate ten dollars to the Frobnitz project for each disk sold." Don't be satisfied with a vague promise, such as "A portion of the profits are donated," since it doesn't give a basis for comparison.

Even a precise fraction "of the profits from this disk" is not very meaningful, since creative accounting and unrelated business decisions can greatly alter what fraction of the sales price counts as profit. If the price you pay is $50, ten percent of the profit is probably less than a dollar; it might be a few cents, or nothing at all.

Some redistributors do development work themselves. This is useful too; but to keep everyone honest, you need to inquire how much they do, and what kind. Some kinds of development make much more long-term difference than others. For example, maintaining a separate version of a program contributes very little; maintaining the standard version of a program for the whole community contributes much. Easy new ports contribute little, since someone else would surely do them; difficult ports such as adding a new CPU to the GNU Compiler Collection contribute more; major new features or packages contribute the most.

By establishing the idea that supporting further development is "the proper thing to do" when distributing free software for a fee, we can assure a steady flow of resources into making more free software.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘`show w`’ and ‘`show c`’; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Option Index

Open64's command line options are indexed here without any initial '`-`' or '`--`'. Where an option has both positive and negative forms (such as '`-f`*option*' and '`-fno-`*option*'), relevant entries in the manual are indexed under the most appropriate form; it may sometimes be useful to look up both forms.

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

Keyword Index

H

I

K

L

M

O

P

R

S

T

U

V

W

X