NAME

```
     ftn - Invokes the Cray Fortran Compiler
```

SYNOPSIS

```
     ftn
     [-A  module_name[, module_name] ...]
     [-b bin_obj_file]
     [-c]
     [-d disable]
     [-D identifier[=value]]
     [-e enable]
     [-f source_form]
     [-F]
     [-g]
     [-G debug_lvl]
     [-h arg]
     [-I incldir]
     [-J dir_name]
     [-K trap=opt[, opt] ...]
     [-l libname]
     [-L ldir]
     [-m msg_lvl]
     [-M msgs]
     [-N col]
     [-o out_file]
     [-O opt[, opt] ...]
     [-p module_site]
     [-Q path]
     [-r list_opt]
     [-R runchk]
     [-rpath ldir]
     [-s size]
     [-S asm_file]
     [-T]
     [-U identifier[, identifier] ...]
     [-v]
     [-V]
     [-Wphase,"opt...",]
     [-x dirlist]
     [-X npes]
     [-Yphase,dirname]
     [--]
     sourcefile [sourcefile ...]
```

IMPLEMENTATION

Cray Linux Environment (CLE)

DESCRIPTION

The ftn command invokes the Cray Fortran Compiler. Typically, the command processes the input files named on the command line and generates a binary object file, and then loads the binary object file and generates the executable file a.out.

When options are specified that are not available, the compiler ignores them and continues with the compilation.

For a more detailed description of the ftn command, see the Cray Fortran Reference Manual. Information contained in this man page may differ from the reference manual. Where the information differs, this man page supersedes the information contained in the reference manual.

The ftn command accepts the following options:

-A module_name[,module_name] ...

Directs the compiler to behave as if you entered a USE module_name statement for each module_namein your Fortran source code. The USE statements are entered in every program unit and interface body in the source file being compiled.

-b bin_obj_file

Disables the link step and saves the binary object file of your program in bin_obj_file.

Only one input file is allowed when the -b bin_obj_file option is specified. If you have more than one input file, use the -c option instead. If only one input file is being processed and neither the -b nor -c option is specified, the binary object file of your program is not saved after the link step is completed.

If both -b bin_obj_file and -c are specified, the link step is disabled and the binary object file is written to bin_obj_file.

Default: disabled.

-c

Disables the link step and saves the binary object file version of your program in file.o, where file is the name of the source file. If there is more than one source file, a file.o is created for each input file specified.

Default: off.

-d disable, -e enable

Disables or enables compiling options. To specify more than one compiling option, enter the options without separators between them; for example, -e aj. disable/enable can be one or more of the following options:

Option	Action
0	Initializes all undefined local stack variables to 0 (zero). If a user variable is of type character, it is initialized to NUL. The variables are initialized upon each execution of the procedure. Default: disabled.
a	Aborts compilation after encountering the first error. Default: disabled.
A	Treat all module variables as PUBLIC. Do not override any explicit PRIVATE statements or attributes. Disabling this option with -dA has the effect of including a PRIVATE statement in the specification part of the module. Default: enabled.
b	If enabled, issue an warning message rather than an error message when the compiler detects a call to a procedure with one or more dummy arguments having the TARGET, VOLATILE or ASYNCHRONOUS attribute and there is not an explicit interface definition. Default: disabled.
B	Generates binary output. If disabled, inhibits all optimization and allows only syntactic and semantic checking. Default: enabled.

c Interface checking: use Cray's system modules to check library calls in a compilation. If you have a procedure with the same name as one in the library, you will get errors, as the compiler does not skip user-specified procedures when performing checks.

Default: disabled.

C Enable/disable some types of standard call site checking. The current Fortran standard requires that the number and types of arguments must agree between the caller and callee. These constraints are enforced in cases where the compiler can detect them, however, specifying -dC disables some of this error-checking, which may be necessary in order to get some older Fortran codes to compile.

Note: If error-checking is disabled, unexpected compile-time or runtime errors may occur.

In addition, the compiler by default attempts to detect situations in which an interface block should be specified but is not. Specifying -dC disables this type of checking as well.

Default: enabled.

d Controls a column-oriented debugging feature when using fixed source form. When enabled, the compiler replaces a D or d character appearing in column 1 of your source with a blank and treats the entire line as a valid source line. This feature is useful if you want to insert PRINT statements as part of your debugging process.

Default: disabled.

D Enables all debugging options. This option is equivalent to specifying the -O0, -G0, -g, -m2, -rl or -R bcdsp options.

Default: disabled.

E The -eE option allows existing declarations to duplicate the declarations contained in a used module. Only existing declarations that declare the function name or generic name in an EXTERNAL or type statement are allowable under this option. Therefore, you do not have to modify the older code by removing the existing declarations. Because the declarations are not removed, the use associated objects duplicates declarations already in the code, which is not standard conforming. However, this option allows the compiler to accept these statements as long as the declarations match the declarations in the module.

Existing declarations of a procedure must match the interface definitions in the module; otherwise an error is generated.

f — Allows the creation of lower-case module .mod file names, in a manner similar to the -em option.

Default: disabled.

F — Controls preprocessor expansion of macros in Fortran source lines.

Default: enabled.

g — Allows branching into the code block for a DO or DO WHILE construct, which may be necessary in order to permit older codes to compile.

Historically, codes used branches out of and into DO constructs. Current Fortran standards prohibit branching into a DO construct from outside of that construct and the compiler issues an error in this situation. Specifying the -eg option will allow codes with these constructs to compile, but performance may suffer as a result.

Default: disabled.

h — Enables support for 8-bit and 16-bit INTEGER and LOGICAL types that use explicit kind or star values.

By default (-eh), data objects declared as

INTEGER(kind=1) or LOGICAL(kind=1) are 8 bits long and objects declared as INTEGER(kind=2) or LOGICAL(kind)=2 are 16 bits long. When this option is disabled (-dh), data objects declared as INTEGER(kind)=1, INTEGER(kind)=2, LOGICAL(kind)=1, or LOGICAL(kind)=2 are 32 bits long.

Note: Vectorization of 8- and 16-bit objects is deferred.

Default: enabled.

I Treat all variables as if an IMPLICIT NONE statement had been specified. Do not override any IMPLICIT statements or explicit type statements. All variables must be typed.

Default: disabled.

j Execute DO loops at least once.

Default: disabled.

m When this option is enabled, the compiler creates .mod files to hold module information for future compiles. When it is disabled, and a module is compiled, the compiler deletes any existing MODULENAME.mod files it finds in the output directory before creating new module information in the .o file.

By default, module files are written to the current working directory. You can use the -J dir_name option to specify an alternate output directory for .mod files only.

Whether this option is enabled or disabled, the search order for satisfying module references in USE statements is as follows:

1. The current working directory.

2. Any directories or files specified with the -p option.

3. Any directories specified with the -I option.

4. Any directories or files specified with the FTN_MODULE_PATH environment variable.

When searching within a directory, the compiler checks all the .mod files first, then the .ofiles, and then the .a files.

Note: The compiler creates modules through the MODULE statement. A module is referenced with the USE statement. All .mod files are named modulename.mod, where modulename is the name of the module specified in the MODULE or USE statement.

Default: disabled.

n	Generates messages to note nonstandard Fortran usage. Default: disabled.
o	Display to stderr the optimization options the compiler used for this compilation. Default: disabled.
P	Performs source preprocessing on Fortran source files, but does not compile. When specified, source code is included by #include directives but not by Fortran INCLUDE lines. Generates file.i, which contains the source code after the preprocessing has been performed and the effects applied to the source program. Default: disabled.
q	Aborts compilation if 100 or more errors are generated. Default: enabled.
Q	Controls whether or not the compiler accepts variable names that begin with a leading underscore (_) character. For example, when -e Q is specified, the compiler accepts _ANT as a

variable name. Enabling this option can cause collisions with system name space; for example, library entry point names.

Default: disabled.

R

Compiles all functions and subroutines as if they contained a RECURSIVE keyword.

Default: disabled.

s

Scale the values of the count and count_rate arguments for the SYSTEM_CLOCK intrinsic function down by a factor of 2**14 (16384) if the storage size of the values of each of the count and count_rate arguments is 32 bits.

Default: enabled.

S

Generates assembly language output and saves it in file.s. When both the -eS and -S asm_file options are specified, the -S asm_file option takes precedence.

Default: disabled.

v

Allocate variables to static storage. These variables are treated as if they had appeared in a SAVE statement. Variables that are explicitly or implicitly defined as automatic variables are not allocated to static storage.

The following types of variables are not allocated to static storage: automatic variables (explicitly or implicitly stated), variables declared with the AUTOMATIC attribute, variables allocated in an ALLOCATE statement, and local variables in explicit recursive procedures. Variables with the ALLOCATABLE attribute remain allocated upon procedure exit, unless explicitly deallocated, but they are not allocated in static memory. Variables in explicit recursive procedures consist of those in functions, in subroutines, and in internal procedures within functions and subroutines that have been defined with the RECURSIVE attribute. The STACK compiler directive overrides this option.

Default: disabled.

w	Enables support for automatic memory allocation for allocatable variables and arrays that are on the left hand side of intrinsic assignment statements. Using this option may degrade runtime performance, even when automatic memory allocation is not needed. It can affect optimizations for a code region containing an assignment to allocatable variables or arrays; for example, by preventing loop fusion for multiple array syntax assignment statements with the same shape. Default: disabled.
y	(Deferred implementation) Adds information into the binary files that enables the linker to find the modules when used in subsequent compiles. The -d y option disables this information. Consequently, subsequent compiles that use these modules must specify the correct information on the linker command line. If the binary files for the Fortran modules are moved prior to the link step, specify -dy. Default: enabled.
z	Initialize all memory allocated by Fortran ALLOCATE statements to zero. This option applies only for the current source file and should be specified for each source file compilation where this behavior is desired. Default: disabled.
Z	Perform source preprocessing and compilation on Fortran source files. When specified, source code is included by both #include directives and Fortran INCLUDE lines. Generates file file.i, which contains the source code after the preprocessing has been performed and the effects applied to the source program. Default: disabled.

-D identifier[=value]
Defines variables used for source preprocessing as if they had been defined by a #define source preprocessing directive. If a value is specified, there can be no spaces on either side of the equal sign. If no value is specified, the default value is 1.

Compare to the -U option.

By default, macros are not expanded in Fortran source statements. Use the -F option to enable macro expansion in Fortran source statements.

-f source_form
Specifies whether the Fortran source file is written in fixed source form or free source form. For source_form, enter free or fixed.

The default is fixed for source files that have a .f or .F suffix. The default is free for source files that have a .f90, .F90, .f95, .F95, .f03, .F03, .f08, .F08, .ftn, or .FTN suffix.

If the file has a .F, .F90, .F95, .F03, .F08, or .FTN suffix, the source preprocessor is invoked.

-F
Macro expansion is now enabled by default and controlled by the -d|e F option. The -F option is obsolete and supported for compatibility with legacy make files.

-g
Provides debugging support identical to specifying the -G0 option.

Default: off.

-G debug_lvl
Controls the tradeoffs between ease of debugging and compiler optimizations. The compiler produces some level of internal debugger information (DWARF) at all times. This DWARF data provides function and source line information to debuggers for tracebacks and breakpoints, as well as type and location information about data variables.

Note: The -g or -G options can be specified on a per-file basis, so that only part of an application pays the price for improved debugging.

debug_lvl	Support
0	All optimizations disabled including floating point optimizations: full DWARF information is available for debugging, but at the cost of a slower and larger executable. Breakpoints can be set at each line. This level of debugging is supported when optimization is disabled; that is, when -O0, -O ipa0, -O scalar0,-O thread0, and -O vector0 are in effect. Implies -h fp0.
1	Partial optimization: most DWARF and at least some optimizations make tracebacks and limited breakpoints available in the debugger. Some scalar optimizations and all loop nest restructuring is disabled, but the source code will be visible and most symbols will be available. This allows block-by-block debugging, with the exception of innermost loops. The executable will be faster than with -g or -G0.
2	Full optimization: with partial DWARF and most optimizations, tracebacks and very limited breakpoints are available in the debugger. The source code will be visible and some symbols will be available. This level allows post-mortem debugging, but local information such as the value of a loop index variable is not necessarily reliable at this level because such information often is carried in registers in optimized code. The executable will be faster and smaller than with -G1.
fast	Compile code for use with Cray fast-track debugging. This option is useful only if used in conjunction with a debugger that supports fast-track debugging. For more information, see the lgdb(1) man page.

-h arg The -h arg option enables you to access various compiler functions. Some of these options duplicate -O arg options; the -h options are provided as a convenience for programmers who mix Fortran and C/C++ code.

[no]acc   Enables or disables the compiler recognition of
          OpenACC accelerator directives.

          Default: acc

[system|default]_alloc
          The -hsystem_alloc option causes the compiler to
          use the native malloc implementation provided by
          the OS. By default, the compiler uses a modified
          malloc implementation which offers better support
          for Cray XE memory needs. This is a link-time
          option.

          Default: default_alloc

[no]add_paren
          The -hadd_paren option automatically adds
          parenthesis to select associative operations
          (+,-,*) to encourage left to right evaluation of
          floating point and complex expressions. Left to
          right evaluation is not required by the language
          standards, but some applications may expect it.

          Default: noadd_paren

[no]align_arrays
          Controls padding of arrays in static data. Some
          statically allocated arrays are aligned and padded
          for better cache behavior. Common block data is
          not affected.

          Default: align_arrays

[no]autoprefetch
          Controls automatic prefetch optimization. Does not
          affect loop_info [no]prefetch directive.

          Default: autoprefetch.

[no]autothread
          The -h [no]autothread option enables or disables
          autothreading.

```
          Default: noautothread

byteswapio
          Forces byte-swapping of all input and output files
          for direct and sequential unformatted I/O.

cachen    Specify the level of automatic cache management to
          be performed, where n is a value from 0 to 3 with
          0 being no cache management and 3 being the most
          aggressive. This is identical to the -O cachen
          option.

          Default: cache2

[no]caf   Enable the compiler to recognize coarray syntax.
          Coarrays are a Fortran 2008 feature that offer a
          method for performing data passing.

          Default: nocaf

cpu=target_system
          Specify the target Cray system on which the
          absolute binary file is to be executed, where
          target_system can be either x86-64, opteron,
          barcelona, shanghai, istanbul, mc8, mc12, or
          interlagos.

          The x86-64 and opteron options produce identical
          output, for use on single- and dual-core systems.
          If you are creating executables for use on a
          system with quad-core processors (either AMD
          Opteron barcelona or shanghai processors), you
          must also have the associated module (either xtpe-
          barcelona or xtpe-shanghai) loaded when compiling
          and linking your code. Likewise, if you are
          creating executables for use on a system with AMD
          Opteron six-core processors (istanbul), eight-core
          processors (mc8), twelve-core processors (mc12),
          or 16-core processors (interlagos), you must have
          the xtpe-istanbul, xtpe-mc8, xtpe-mc12, or xtpe-
          interlagos module loaded when compiling and
          linking your code. If one of these modules is
          loaded, the default target_system changes to the
          corresponding cpu target.
          If target_system is set during compilation of any
          source file, it must be set to the same target
          during linking and loading.

          The target system may also be specified using the
          CRAY_PE_TARGET environment variable.

          Default: x86-64.

display_opt
          Display the compiler optimization settings
          currently in force. This option is identical to
          the -eo option.

[no]dwarf Controls whether DWARF debugging information is
          generated during compilation.

          Default: dwarf.

dynamic   Directs the compiler driver to link dynamic
          libraries at runtime. This option is used to
          create dynamically linked executable files and may
          not be used with the -h static or -h shared
          options. Note that the preferred invocation is to
          call the generic ftn command with the -dynamic
          option, rather than using this compiler specific
          option. See the ftn(1) man page.

flex_mp=level
          Controls the aggressiveness of optimizations which
          may affect floating point and complex
          repeatability when application requirements
          require identical results when varying the number
          of ranks or threads.

          -hflex_mp=intolerant has the highest probability
          of repeatable results, but also has the highest
          performance penalty. -hflex_mp=conservative uses
          more aggressive optimization and yields higher
          performance than -hflex_mp=intolerant, but results
          may not be sufficiently repeatable for some
          applications. -hflex_mp=tolerant uses most
          aggressive optimization and yields highest
          performance, but results may not be sufficiently
          repeatable for some applications.
```

fpn
: Controls the level of floating point optimizations, where n is a value between 0 and 3, with 0 giving the compiler minimum freedom to optimize floating point operations and 3 giving it maximum freedom. The higher the level, the less the floating point values conform to the IEEE standard.

  When -hfp[0,1] is specified, it also has the effect of setting -hfp_trap.

  Default: fp2.

[no]fp_trap
: Controls whether the compiler generates code compatible with floating point traps being enabled.

  Default: fp_trap, if traps are enabled using the -K trap option, or if -Ofp[0,1] is in effect. Otherwise, the default is nofp_trap.

[no]func_trace
: The -h func_trace option is for use only with CrayPat (Cray performance analysis tool). If this option is specified, the compiler inserts CrayPat entry points into each function in the compiled source file. The names of the entry points are __pat_tp_func_entry and __pat_tp_func_return.

  These are resolved by CrayPat when the program is instrumented using the pat_build command. When the instrumented program is executed and it encounters either of these entry points, CrayPat captures the address of the current function and its return address.

  Default: nofunc_trace

keepfiles
: The -h keepfiles option prevents the removal of the object (.o) and temporary assembly (.s) files after an executable is created. Normally, the compiler automatically removes these files after linking them to create an executable. Since the original object files are required in order to instrument a program for performance analysis, if

```
          you plan to use CrayPat to conduct performance
          analysis experiments, you can use this option to
          preserve the object files.

loop_trips=[tiny|small|medium|large|huge]
          Specifies runtime loop trip counts for all loops
          in a compiled source file. This information is
          used to better tune optimizations to the runtime
          characteristics of the application.

mpin      Enables or disables optimization around a selected
          subset of MPI library calls. mpi0 disables this
          option.

          Default: mpi1(on)

[no]msgs  Controls whether messages describing optimizations
          performed are written to stderr.

          Similar information in a more-readable format can
          be obtained by using the -rm option instead.

          This option is identical to the -O [no]msgs
          option.

          Default: nomsgs

[no]negmsgs
          Controls whether messages explaining why
          optimizations such as vectorization or inlining
          did not occur are written to stderr.

          The -h negmsgs option enables the -h msgs option.
          The -rm option enables the -h negmsgs option.

          This option is identical to the -O [no]negmsgs
          option.

          Default: nonegmsgs

network=nic
          Specify the target machine's interconnection
          attribute. The supported value is gemini.

[no]omp   Enable or disable compiler recognition of OpenMP
          directives. Using -h noomp is similar to the -h
          thread0 option, in that it disables OpenMP, but
          unlike -h thread0 it does not affect
          autothreading. The -h noomp option is identical to
          the -O [no]omp option.

          Default: omp

[no]omp_acc
          Enables or disables the compiler recognition of
          OpenMP accelerator directives.

          Default: omp_acc

[no]omp_trace
          Enable or disable the insertion of CrayPat OpenMP
          tracing calls.

          Default: noomp_trace.

page_align_allocate
          The -h page_align_allocate option directs the
          compiler to force allocations of arrays larger
          than the memory page size to be aligned on a page
          boundary. This option affects only the ALLOCATE
          statements of the current source file; therefore
          it must be specified for each source file where
          this behavior is desired. Using this option can
          improve DIRECTIO performance.

pic, PIC  Generate position independent code (PIC), which
          allows a virtual address change from one process
          to another, as is necessary in the case of shared,
          dynamically linked objects. The virtual addresses
          of the instructions and data in PIC code are not
          known until dynamic link time.

pl=program_library
          Create and use a persistent repository of compiler
          information specified by program_library. When
          used with -h wp, this option provides application-
          wide, cross-file, automatic inlining. See -h wp.

          The program_library repository is implemented as a
          directory and the information contained in program
          library is built up with each compiler invocation.
          Any compilation that does not have the -h pl
```

```
                    option will not add information to this
                    repository.

                    Because of the persistence of program_library, it
                    is the user's responsibility to manage it. For
                    example, rm -r program_library might be added to
                    the make clean target in an application makefile.
                    Because program_library is a directory, use rm -r
                    to remove it.

                    If an application makefile works by creating files
                    in multiple directories during a single build, the
                    program_library should be an absolute path,
                    otherwise multiple and incomplete program library
                    repositories will be created. For example, avoid
                    -hpl=./PL.1 and use -hpl=/fullpath/builddir/PL.1
                    instead.

          profile_generate
                    Enable instrumenting of source code for CrayPat
                    profile-guided optimization. For more information,
                    see the intro_craypat(1) and pat_build(1) man
                    pages.

          [no]second_underscore
                    Control the way in which external names are
                    generated. By default, the compiler generates
                    external names in lower case and adds one trailing
                    underscore. This behavior matches the PGI Fortran
                    compiler's external behavior. If this option is
                    enabled, the compiler adds a second trailing
                    underscore if the original external name has any
                    underscores in it. This behavior matches the GNU
                    compiler's external behavior.

                    Default: nosecond_underscore.

          shared    Creates a library which may be dynamically linked
                    at runtime. Note that the preferred invocation is
                    to call the generic ftn command with the -shared
                    option, rather than using this compiler specific
                    option. See the ftn(1) man page.

          static    Directs the linker to use the static version of
                    the libraries, not the dynamic version of the
                    libraries, to create an executable file. Note that
```

```
              the preferred invocation is to call the generic
              ftn command with the -static option. See the
              ftn(1) man page.

    threadn   Control the compilation and optimization of OpenMP
              directives, where n is a value from 0 to 3 with 0
              being off and 3 specifying the most aggressive
              optimization. This option is identical to the -O
              threadn option.

              Default: thread2.

    wp        Enables the whole program mode. This option causes
              the compiler backend (IPA, optimizer,
              codegenerator) to be invoked at application link
              time, enabling whole program automatic
              inlining/cloning and future whole program
              interprocedural analysis (IPA) optimizations.
              Since the -hwp option provides automatic
              application-wide inlining, the -Oipafrom option is
              no longer needed for cross-file inlining. Requires
              that -h pl=program_library is also specified.

              The options -h pl=program_library and -hwp should
              be specified on all compiler invocations and on
              the compiler link invocation. Since -h wp delays
              the compiler optimization step until link time, -c
              compiles will take less time and the link step
              will take longer. Normally, this is just a time
              shift from one build phase to another with roughly
              the same overall compile time. In some cases
              increased inlining may cause an increase in
              overall compile time. Using -h wp allows the
              compiler backend to be invoked in parallel during
              a build. Setting the environment variable NPROC
              controls the number of concurrent compiler backend
              invocations and this parallelism may reduce
              overall compile time.

    zero      Initializes all undefined local stack variables to
              0 (zero). If a user variable is of type character,
              it is initialized to NUL. The variables are
              initialized upon each execution of the procedure.
              This option is identical to the -e0 option.

              Default: disabled.
```

-I incldir

Specifies a directory to be searched for files named in INCLUDE lines and #include directives. You must specify an -I option for each directory you want searched. Directories can be specified in incldir as full pathnames or as pathnames relative to the working directory.

If no -I is specified, only the working directory and system directories are searched.

-J dir_name

Specifies the directory to which the file.mod files are written when -e m is specified on the command line.

The compiler automatically searches the dir_name directory for modules to satisfy USE statements. An error is issed if the -em option is not specified when the -J option is used.

By default, the files are written to the current working directory.

-K trap=opt[,opt] ...

Enable traps for the specified exceptions. By default, no exceptions are trapped. Enabling traps using this option also has the effect of setting -h fp_trap.

If the specified options contradict each other, the last option has priority. For example, -Ktrap=none,fp is equivalent to -Ktrap=fp.

This option is processed only at link time and affects the entire program; it is not processed when compiling subprograms. Therefore, traps may be set using this command line option at the beginning of execution of the main program only. The program may subsequently change these settings by calling intrinsic or library procedures. Use of this option may require the specification of -hfp_trap when compiling other files of the application.

opt	Exceptions
denorm	Trap on denormalized operands.
divz	Trap on divide-by-zero.

fp	Trap on divz, inv, or ovf exceptions.
inexact	Trap on inexact result (i.e. rounded result). Enabling traps for inexact results is not recommended.
inv	Trap on invalid operation.
none	Disables all traps (default).
ovf	Trap on overflow (i.e. the result of an operation is too large to be represented).
unf	Trap on underflow (i.e. the result of an operation is too small to be represented).

-l libname

Directs the compiler driver to search for the specified object library file when linking an executable. To request more than one library file, specify multiple -l options.

When statically linking, the compiler driver searches for libraries by prepending ldir/lib on the front of libname and appending .a on the end of it, for each ldir that has been specified by using the -L option. It uses the first file it finds.

When dynamically linking, the library search process is similar to the static case, with a few differences. The compiler driver searches for libraries by prepending ldir/lib on the front of libname and appending .so on the end of it, for each ldir that has been specified by using the -L option. If a matching .so is not found, the compiler driver replaces .so with .a and repeats the process from the beginning. It uses the first file it finds.

There is no search order dependency for libraries.

If you specify personal libraries by using the -l command line option, those libraries are added before the default CCE library list.

For example, when the following command line is issued, the linker looks for a library named libmylib.a (following the naming convention) and adds it to the top of the list of default libraries.

```
% ftn -l mylib target.f
```

-L ldir

Changes the -l option search algorithm to look for library files in directory ldir. To request more than one library directory, specify multiple -L options.

Note: Multiple -L options are treated cumulatively as if all ldir arguments appeared on one -L option preceding all -l options. Therefore, do not attempt to link functions of the same name from different libraries through the use of alternating -L and -l options.

The compiler driver searches for library files in directory ldir before searching the default directories: /opt/ctl/libs and /lib.

For example, when statically linking, if -L ../mylib, -L /loclib, and -l m are specified, the compiler driver searches for the following files and uses the first one found:

```
../mylibs/libm.a
/loclib/libm.a
/opt/ctl/libs/libm.a
/lib/libm.a
```

-m msg_lvl

Specifies the minimum compiler message levels to enable. The following list shows the integers to specify in order to generate each type of message and which messages are generated by default:

msg_lvl	Message Types Enabled
0	Error, Warning, Caution, Note, and Comment
1	Error, Warning, Caution, and Note

2	Error, Warning, and Caution
3	Error and Warning (default)
4	Error

You can use the explain(1) command to view a message explanation.

-M msgs The -M msgs option suppresses messages at the Warning, Caution, Note, and Comment levels and can change the default message severity to an Error or a Warning level. You cannot suppress or alter the severity of Error-level messages with this option.

To suppress messages, specify one or more integer numbers that correspond to the Cray Fortran Compiler messages you want to suppress. To specify more than one message number, specify a comma (but no spaces) between the message numbers. For example, -M 110,300 suppresses messages 110 and 300.

To change a message's severity to an Error level or a Warning level, specify an E (for Error) or a W (for Warning) and then the number of the message. For example, consider the following option:

```
-M 300,E600,W400
```

This specification results in the following:

¬∑ Message 300 is disabled and is not issued, provided that it is not an Error-level message by default. Error-level messages cannot be suppressed and cannot have their severity downgraded.

¬∑ Message 600 is issued as an Error-level message, regardless of its default severity.

¬∑ Message 400 is issued as a Warning-level message, provided that is it not an Error-level message by default.

-N col Specifies the line width, in columns, for fixed- or free-format source lines. For fixed form sources, use one of the following values for col to specify the maximum number of columns per line:

¬∑ 72

¬∑ 80

¬∑ 132

¬∑ 255

For free form sources, col can be set to 132 or 255.

By default, lines are 72 characters wide for fixed-format sources and 255 characters wide for free-form sources.

-O opt[,opt] ...

Specifies optimization features. The opt values 0, 1, 2, and 3 enable you to specify increasing general levels of optimization. The other opt values enable you to select specific optimization features.

The -O 1, -O 2, and -O 3 specifications do not directly correspond to the numeric optimization levels for scalar optimization and vectorization. For example, specifying -O 3 does not necessarily enable scalar3 and vector3. Cray reserves the right to alter the specific optimizations performed at these levels from release to release. You can use the -e o option or the ftnlx command to display the optimization options used during compilation.

The valid opt values are:

opt	Optimization Provided
-O 0	Disables all optimizations including floating point optimizations. Implies -h fp0.
-O 1, -O 2, -O 3	Default: 2.
[no]aggress	Cause the compiler to treat a program unit (for example, a subroutine or function) as a single optimization region. Doing so can improve the optimization of large program units but also increases compile time and size.

Default: noaggress.

[no]autoprefetch
Controls automatic prefetch optimization. Does not affect loop_info [no]prefetch directive.

Default: autoprefetch.

cachen Specify the level of automatic cache management, where n can be one of the following values:

0	Specifies no automatic cache management; all memory references are allocated to cache. Both automatic cache blocking and manual cache blocking (by use of the BLOCKABLE directive) are shut off. Characteristics include low compile time. This option is compatible with all optimization levels.
1	Specifies conservative automatic cache management. Characteristics include moderate compile time. Symbols are placed in the cache when the possibility of cache reuse exists and the predicted cache footprint of the symbol in isolation is small enough to experience reuse.
2	Specifies moderately aggressive automatic cache management. Characteristics include moderate compile time. Symbols are placed in the cache when the possibility of cache reuse exists and the predicted state of the cache model is such that the symbol will be reused.
3	Specifies aggressive automatic cache management. Characteristics include potentially high compile time. Symbols are placed in the cache when the possibility of cache reuse exists and the allocation of the symbol to the cache is predicted to increase the number of cache hits.

```
fpn       Controls the level of floating point
          optimizations, where n is a value between 0 and 3,
          with 0 giving the compiler minimum freedom to
          optimize floating point operations and 3 giving it
          maximum freedom. The higher the level, the less
          the floating point values conform to the IEEE
          standard.

          When -hfp[0,1] is specified, it also has the
          effect of setting -hfp_trap.

          Default: fp2.

fusionn   Control loop fusion globally and changes the
          assertiveness of the FUSION directive.

          Loop fusion can improve the performance of loops.
          although in some rare cases it may degrade overall
          performance.

          The n argument enables you to turn loop fusion on
          or off and determine where fusion should occur. It
          also affects the assertiveness of the FUSION
          directive. n can be one of the following values:

          0         No fusion (ignore all FUSION directives
                    and do not attempt to fuse other loops)

          1         Attempt to fuse loops that are marked by
                    the FUSION directive.

          2         Attempt to fuse all loops (includes
                    array syntax implied loops), except
                    those marked with the NOFUSION
                    directive.

          Default: fusion2.

inlinelib (Deferred implementation) Attempt inlining of
          those Cray scientific library routines that are
          available for inlining. For a report of what was
          inlined or not, see the -O msg,negmsgs option.

ipan      Control level of interprocedural analysis (IPA)
          which implies the control over the level of
```

automatic inlining and cloning.

Inlining is the process of replacing a user procedure call with the procedure definition itself. This saves subprogram call overhead and may allow better optimization of the inlined code. If all calls within a loop are inlined, the loop becomes a candidate for parallelization.

Cloning is a situation in which a procedure is duplicated with modifications such that it will run more efficiently. For example, the compiler will clone a procedure for a specific call site when there are constant actual arguments present in that call site. When the clone is made, the dummy arguments are replaced with the constant actual arguments, and the original call to the procedure is replaced with a call to the duplicate copy.

When -O ipan is used alone, the candidates for expansion are all those functions that are present in the input file to the compile step. If -O ipan is used in conjunction with -O ipafrom=source, the candidates for expansion are those functions present in source.

The valid values for n are:

0	All inlining and cloning is disabled. All inlining and cloning compiler directives are ignored.
1	Directive inlining. Inlining is attempted for call sites and routines that are under the control of an inlining compiler directive. Cloning is not enabled and cloning directives are ignored.
2	Call nest inlining. Inline a call nest to an arbitrary depth as long as the nest does not exceed some compiler-determined threshold. A call nest can be a leaf routine. The expansion of the call nest must yield straight-line code

```
                    (code containing no external calls) for
                    any expansion to occur. The call site is
                    said to "flatten" when there are no
                    calls present in the expanded code. The
                    call site must reside within the body of
                    a loop for expansion to be attempted.
                    Cloning is not enabled and cloning
                    directives are ignored.

          3         Constant actual argument inlining and
                    tiny routine inlining. Default level for
                    inlining. This includes levels 1 and 2,
                    plus any call site that contains a
                    constant actual argument. Additionally,
                    any call nest (regardless of location)
                    that is below some small compiler-
                    determined threshold will be inlined
                    provided that call nest completely
                    flattens. Cloning is not enabled and
                    cloning directives are ignored.

          4         Cloning. This includes levels 1, 2, and
                    3, plus routine cloning is attempted if
                    inlining fails at a given call site.

          5         Aggressive interprocedural analysis
                    (IPA). Includes levels 1, 2, 3, and 4.

ipafrom=source[:source] ...
          Explicitly indicate the procedures to consider for
          inline expansion.

          The source arguments identify each file or
          directory that contains the routines to consider
          for inlining. Whenever a call is encountered in
          the input program that matches a routine in
          source, inlining is attempted for that call site.

          Note:  Blank spaces are not allowed on either side
          of the equal sign.

          All inlining directives are recognized with
          explicit inlining.

          Note that the routines in source are not actually
          linked with the final program. They are simply
```

templates for the inliner. To have a routine contained in source linked with the program, you must include it in an input file to the compilation.

The following source arguments are supported.

Fortran source files

The routines in Fortran source files are candidates for inline expansion and must contain error-free code. Source files that are acceptable for inlining are files that have one of the following extensions

¬∑ .f

¬∑ .F

¬∑ .f90

¬∑ .F90

¬∑ .f95

¬∑ .F95

¬∑ .f03

¬∑ .F03

¬∑ .f08

¬∑ .F08

¬∑ .ftn

¬∑ .FTN

Module files

When compiling with -em and -Omodinline in effect, the precompiled module information is written to modulename* .mod. The compiler writes a modulename* .mod file for each module; modulename is created by taking the name of the module

```
                    and, if necessary, converting it to
                    uppercase.

          Directories
                    A directory containing any of the
                    Fortran source of Module files described
                    above.

[no]modinline
          Prepare module procedures so they can be inlined
          by directing the compiler to create templates for
          module procedures encountered in a module. These
          templates are attached to file.o or modulename*
          .mod. The files that contain these inlinable
          templates can be saved and used later to inline
          call sites within a program being compiled.

          When -e m is in effect, module information is
          stored in modname.mod. The compiler writes a
          modulename.mod file for each module; modulename is
          created by taking the name of the module and, if
          necessary, converting it to uppercase.

          The process of inlining module procedures requires
          only that file.o or modulename.mod be available
          during compilation through the typical module
          processing mechanism. The USE statement makes the
          templates available to the inliner. You do not
          need to specify the file.o or modulename.mod with
          the -O ipafrom option.

          When -O modinline is specified, the MODINLINE and
          NOMODINLINE directives are recognized. Using the
          -O modinline option increases the size of file.o.

          To ensure that file.o is not removed, specify this
          option in conjunction with the -c option.

          Default: modinline

[no]msgs  Cause the compiler to write optimization messages
          to stderr.

          Similar information in a more-readable format can
          be obtained by using the -rm option instead.
          Specifying the -rm option enables -O msgs.
          Default: nomsgs

[no]negmsgs
          Cause the compiler to generate messages to stderr
          that indicate why optimizations such as
          vectorization or inlining did not occur in a given
          instance.

          The -O negmsgs option enables the -O msgs option.
          The -rm option enables the -O negmsgs option.

          Default: nonegmsgs

nointerchange
          Inhibit the compiler's attempts to interchange
          loops. Interchanging loops by having the compiler
          replace an inner loop with an outer loop can
          increase performance. The compiler performs this
          optimization by default.

          Specifying the -O nointerchange option is
          equivalent to specifying a NOINTERCHANGE directive
          prior to every loop. To disable loop interchange
          on individual loops, use the NOINTERCHANGE
          directive.

[no]omp   Enable or disable compiler recognition of OpenMP
          directives. Using -O noomp is similar to the -O
          thread0 option, in that it disables OpenMP, but
          unlike -O thread0 it does not affect
          autothreading. The -O noomp option is identical to
          the -h [no]omp option.

          Default: omp

[no]overindex
          Assert that there are no array subscripts which
          index a dimension of an array that are outside the
          declared bounds of that dimension. Short loop code
          generation occurs when the extent does not exceed
          the maximum vector length of the machine.
          Specifying -O overindex declares that the program
          contains code that makes array references with
          subscripts that exceed the defined extents. This
          prevents the compiler from performing the short
          loop optimizations.

          Default: nooverindex

[no]pattern
          Enables pattern matching for library substitution.
          The pattern matching feature searches your code
          for specific code patterns and replaces them with
          calls to highly optimized routines.

          The -O pattern option is enabled only for
          optimization levels -O 2, -O vector2 or higher;
          there is no way to force pattern matching for
          lower levels.

          Specifying -O nopattern disables pattern matching
          and causes the compiler to ignore the PATTERN and
          NOPATTERN directives.

          Default: pattern

scalarn   Specifies the level of scalar optimization, where
          n can be one of the following levels:

          0         Disables scalar optimization.

          1         Specifies conservative scalar
                    optimization.

          2         Specifies moderate scalar optimization.
                    This is the default.

          3         Specifies aggressive scalar
                    optimization.

shortcircuitn
          Specifies various levels of short circuit
          evaluation, which is an optimization in which the
          compiler analyzes all or part of a logical
          expression based on the results of a preliminary
          analysis. When enabled, the compiler attempts
          short circuit evaluation of logical expressions
          that are used in IF statement scalar logical
          expressions. This evaluation is performed on the
          .AND. and .OR. operator. n can be one of the
          following levels:
```

0	Disables short circuiting of IF and ELSEIF statement logical conditions.
1	Specifies short circuiting of IF and ELSEIF logical conditions only when a PRESENT, ALLOCATED, or ASSOCIATED intrinsic procedure is in the condition.
2	Specifies short circuiting of IF and ELSEIF logical conditions, and it is done left to right. This is the default for x86-64.
3	Specifies short circuiting of IF and ELSEIF logical conditions. It is an attempt to avoid making function calls. If either the left or right operand to .AND. and .OR. operators contain function calls, short circuit evaluation is performed. This is the default for target cpus other than x86-64.

threadn   Control the compilation and optimization of OpenMP directives, where n is a value from 0 to 3 with 0 being off and 3 specifying the most aggressive optimization.

The valid values for n are:

0	No autothreading or OpenMP threading. The -O thread0 option is similar to -O noomp, but -O noomp disables OpenMP only and does not affect autothreading.
1	Specifies strict compliance with the OpenMP standard for directive compilation. Strict compliance is defined as no extra optimizations in or around OpenMP constructs. In other words, the compiler performs only the requested optimizations.
2	OpenMP parallel regions are subjected to some optimizations; that is, some parallel region expansion. Parallel

region expansion is an optimization that merges two adjacent parallel regions in a compilation unit into a single parallel region.

3	Full optimization: loop restructuring, including modifying iteration space for static schedules (breaking standard compliance). Reduction results may not be repeatable.

Default: -O thread2

unrolln The -O unrolln option globally controls loop unrolling and changes the assertiveness of the UNROLL directive. By default, the compiler attempts to unroll all loops, unless the NOUNROLL directive is specified for a loop. Generally, unrolling loops increases single processor performance at the cost of increased compile time and code size.

The n argument enables you to turn loop unrolling on or off and determine where unrolling should occur. It also affects the assertiveness of the UNROLL directive. Use one of these values for n:

0	No unrolling (ignore all UNROLL directives and do not attempt to unroll other loops)
1	Attempt to unroll loops that are marked by the UNROLL directive.
2	Attempt to unroll all loops (includes array syntax implied loops), except those marked with the NOUNROLL directive. This is the default.

Default: unroll2.

vectorn Specifies the level of automatic vectorizing to be performed. Vectorization results in dramatic performance improvements with a small increase in object code size. Vectorization directives are unaffected by this option.

Default: 2.

0	Minimal automatic vectorization. Characteristics include low compile time and small compile size. This option is compatible with all scalar optimization levels. The compiler will still vectorize array syntax in order to allow full source level debugging with reasonable performance. When this option is specified in conjunction with -hfp0 or -hfp1, then array syntax containing associative floating point or complex operations will not be vectorized.
1	Conservative vectorization. The -h vector1 option is compatible with -h scalar1, -h scalar2, and -h scalar3.
2	Moderate vectorization. Loop nests are restructured. The -h vector2 option is compatible with -h scalar2 or -h scalar3.
3	Aggressive vectorization.

[no]zeroinc

Cause the compiler to assume that a constant increment variable (CIV) can be incremented by zero. A CIV is a variable that is incremented only by a loop invariant value. For example, in a loop with variable J, the statement J = J + K, where K can be equal to zero, J is a CIV. -O zeroinc can cause less strength reduction to occur in loops that have variable increments.

Default: nozeroinc

-o out_file

Override the default executable file name, a.out, with the name specified in the out_file argument.

If both the -o out_file and -c options are specified, the link step is disabled and the binary file is written to out_file.

-p module_site[, module_site]

Specify where to look for Fortran modules to satisfy USE statements. The module_site argument specifies the name of a file or directory to search for modules. The module_site specified can be a .mod file, .o (object) file, .a (archive) file, or a directory.

By default, module files are written to the current working directory. Alternatively, you can use the -J dir_name option during compilation to specify an alternate output directory for .mod files only. The compiler will search for modules stored in the directories you specified using the -J dir_name option for the current compilation automatically; you do not need to use the -p option explicitly to make the compiler do this.

The search order for satisfying modules references in USE statements is as follows:

1. The current working directory (or -J dir_name directory, if specified).

2. Any directories or files specified with the -p option.

3. Any directories specified with the -I option.

4. Any directories or files specified with the FTN_MODULE_PATH environment variable.

When searching within a directory, the compiler first searches the .mod files, then the .o files, then the .a files, and then the directories, in the order specified.

-Q path

Specifies the directory to contain all saved nontemporary files from this compilation (for example, all .o and .mod files). Specific file types (such as .o files) are saved to a different directory if the -b, -J, -o, or -S options are used.

By default, this option is disabled and the compiler puts all nontemporary files in the current working directory.

-r list_opt

Produces a listing file. The list_opt arguments are as follows:

Note: Arguments a, c, l, m, o, s, and x invoke the ftnlx(1) command.

a	Includes all reports in the listing (including source, cross references, options, lint, loopmarks, common block, and options used during compilation).
c	Listing includes a COMMON block report (lists all common blocks and members of each block).
d	Decompiles (translates) the intermediate representation of the compiler into listings that resemble the format of the source code. You can use these files to examine the restructuring and optimization changes made by the compiler, which can lead to insights about changes you can make to your Fortran source to improve its performance. The compiler produces two decompilation listing files, with these extensions, per source file specified on the command line: .opt and .cg.
e	Expands included files in the source listing. This option is off by default.
l	Lists source code and includes lint style checking. The listing includes the COMMON block report (see the -r c option for more information about the COMMON block report).
m	Produces a source listing with loopmark information. To provide a more complete report, this option automatically enables the -O negmsg option to show why loops were not optimized. If you do not require this information, use the -O nonegmsg option on the same command line.
o	Show all options used by the compiler during compilation.
s	Lists source code.
T	Retains file.T after processing rather than deleting it.

```
          x         Produces a cross-reference listing.

-R runchk Specifies any of a group of runtime checks for your program.
          To specify more than one type of checking, specify
          consecutive runchk arguments, as follows: -R bs.

          runchk can be one or more of the following suboptions:

          b         Enables checking of array bounds. Bounds checking
                    is not performed on arrays dimensioned as (1).
                    Enables -Ooverindex.

          c         Enables conformance checking of array operands in
                    array expressions.

          d         Enables a run time check for the !dir$ collapse
                    directive and checks the validity of the
                    loop_info, shortloop, and shortloop128 count
                    information.

          p         Generates run time code to check the association
                    or allocation status of referenced POINTER
                    variables, ALLOCATABLE arrays, or assumed-shape
                    arrays.

          s         Enables checking of character substring bounds.

          By default, no runtime checks are performed.

rpath ldir
          The -rpath ldir option changes the run time library search
          algorithm to look for files in directory ldir. To request
          more than one library directory, specify multiple -rpath
          options. Note that a library may be found at link time with
          an -L option, but may not be found at run time if a
          corresponding -rpath option was not supplied on the link
          line. Also note that the compiler driver does not pass the
          -rpath option to the linker. You must explicitly specify -Wl
          when using this option.

          At link time, all ldir arguments are added to the
          executable. The dynamic linker will search these paths first
          for shared dynamic libraries at run time, with one
          exception. The Linux environment variable LD_LIBRARY_PATH
          precedes all other search paths for shared dynamically
```

linked libraries. The use of LD_LIBRARY_PATH is discouraged.

Caution:

Caution should be used when setting LD_LIBRARY_PATH. Doing so will change the shared dynamically linked library search paths for all executable files in your environment.

-s size   The -s size option allows you to modify the sizes of variables, literal constants, and intrinsic function results declared as type REAL, INTEGER, LOGICAL, COMPLEX, DOUBLE COMPLEX, or DOUBLE PRECISION. Use one of these for size:

size	Action
byte_pointer	Applies a byte scaling factor to integers used in pointer arithmetic involving Cray pointers. That is, Cray pointers are moved on byte instead of word boundaries.
default32	Adjusts the data size of default types as follows: ¬∑ 32 bits: REAL, INTEGER, LOGICAL ¬∑ 64 bits: COMPLEX, DOUBLE PRECISION ¬∑ 128 bits: DOUBLE COMPLEX Note: The data sizes of integers and logicals that use explicit kind and star values are not affected by this option. However, they are affected by the -e h option.
default64	Adjust the data size of default types as follows: ¬∑ 64 bits: REAL, INTEGER, LOGICAL ¬∑ 64 bits: DOUBLE PRECISION (implied -dp) ¬∑ 128 bits: COMPLEX ¬∑ 128 bits: DOUBLE COMPLEX (implied -dp)

If you used the -s default64 at compile time, you must also specify this option when invoking the ftn command.

Note: The data sizes of integers and logicals that use explicit kind and star values are not affected by this option. However, they are affected by the -eh option.

integer32 Adjusts the default data size of default integers and logicals to 32 bits.

integer64 Adjusts the default data size of default integers and logicals to 64 bits.

real32 Adjusts the default data size of default real types as follows:

¬∑ 32 bits: REAL

¬∑ 64 bits: COMPLEX and DOUBLE PRECISION

¬∑ 128 bits: DOUBLE COMPLEX

real64 Adjusts the default data size of default real types as follows:

¬∑ 64 bits: REAL

¬∑ 64 bits: DOUBLE PRECISION (implied -dp)

¬∑ 128 bits: COMPLEX

¬∑ 128 bits: DOUBLE PRECISION (implied -dp)

word_pointer

Applies a word scaling factor to integers used in pointer arithmetic involving Cray pointers. That is, Cray pointers are moved on word instead of byte boundaries.

The default data size options (for example, -s default64) option does not affect the size of data that explicitly

```
          declare the size of the data (for example, REAL(KIND=4) R.

          Note:  REAL(KIND=16) and COMPLEX(KIND=16) are not supported.

-S asm_file
          Specifies the assembly language output file name. This
          option overrides the -e S and -b bin_obj_file options.

          By default, this option is off.

-T        Disables the compiler but displays all options currently in
          effect.

          By default, this option is off.

-U identifier[,identifier] ...
          The -U identifier [,identifier] ... option undefines
          variables used for source preprocessing. This option removes
          the initial definition of a predefined macro or sets a user
          predefined macro to an undefined state.

          The -D identifier [=value] option defines variables used for
          source preprocessing. If both -D and -U are used for the
          same identifier, in any order, the identifier is undefined.

          This option is ignored unless one of the following
          conditions is true:



          ¬∑  The Fortran input source file is specified as either
             file.F, file.F90, file.F95, file.F03, file.F08, file.FTN.

          ¬∑  The -e P or -e Z options have been specified.

-v        Prints information about each compilation phase to the
          standard error file (stderr). The information contains what
          the compiler, lister, and linker is doing and what it is
          calling.

          By default, this option is off.

-V        Directs each compilation phase to send a message containing
          version information to the standard error file (stderr).
          Unlike all other command line options, you can specify this
          option without specifying an input file name; that is,
```

specifying ftn -V is valid.

By default, this option is off.

-Wa,"assembler_opt"

The -Wa,"assembler_opt" option passes assembler_opt directly to the assembler. For example, -Wa,"-h" passes the -h option directly the as command, directing it to enable all pseudos, regardless of location field name. This option is meaningful to the system only when file.s is specified as an input file on the command line. For more information about assembler options, see the as(1) man page.

-Wr,"lister_opt"

The -Wr,"lister_opt" option passes lister_opt directly to the ftnlx command. For example, specifying -Wr,"-o cfile.o" passes the argument cfile.o directly to the ftnlx command's -o option; this directs ftnlx to override the default output listing and put the output file in cfile.o. If you specify the -Wr,"lister_opt" option, you must specify the -r list_opt option. For more information about options, see the ftnlx man page.

-x dirlist

Disables specified directives or specified classes of directives. If specifying a multiword directive, either enclose the directive name in quotation marks or remove the spaces between the words in the directive's name. dirlist can be one of the following options:

acc	All OpenACC API directives.
all	All compiler and OpenMP Fortran API directives.
dir	All compiler directives.
directive	One or more compiler directives. If specifying more than one, separate them with commas, as follows: -x INLINEALWAYS,"NO SIDE EFFECTS",BOUNDS.
omp	All OpenMP Fortran API directives, except accelerator directives.
conditional_omp	All C$ and !$ conditional compilation lines.

By default, no directives or specified classes of directives are disabled.

-X npes   Specify the number of processing elements (PEs) that will be specified at job launch. The value for npes can range from 1 through 2**31 - 1 inclusive on Cray XE systems.

If -X is specified, the user must invoke aprun -n npes using the same value for npes. Otherwise, a run time error results.

By default, the compiler does not specify the number of processors.

-Yphase,dirname

Specifies a new directory (dirname) from which the designated phase should be executed. phase can be one or more of the following values:

Table 1. -Yphase Definitions

phase	System Phase	Command
0	Compiler	ftn
a	Assembler	as

--   Signifies the end of options. After this symbol, specify the files to be processed.

sourcefile [sourcefile ...]

Fortran source files to be processed, where sourcefile is one or more of the following:

¬∑ file.f

¬∑ file.F

¬∑ file.f90

¬∑ file.F90

¬∑ file.f95

¬∑ file.F95

¬∑ file.f03

¬∑ file.F03

¬∑ file.f08

¬∑ file.F08

¬∑ file.ftn

¬∑ file.FTN

Files ending in .o and .s are also accepted. By default, several files are created during processing. The Cray Fortran Compiler adds a suffix to the file portion of the file name and places the files it creates in your working directory.

ENVIRONMENT VARIABLES

The Cray Fortran Compiler recognizes these compile-time environment variables (for OpenMP environment variables, see Cray Fortran Reference Manual):

CRAY_FTN_OPTIONS
: Specifies additional options to attach to the command line.

CRAY_PE_TARGET
: Specifies the target_system for compilation. The command line option -h cpu=target_system takes precedence over the CRAY_PE_TARGET setting. The currently acceptable values for CRAY_PE_TARGET are x86-64, opteron, barcelona, shanghai, istanbul, mc8, mc12, or interlagos.

The x86-64 and opteron options produce identical output, for use on single- and dual-core systems. If you are creating executables for use on a barcelona or shanghai (quad-core), istanbul (six-core), mc8 (8-core), mc12 (12-core), or interlagos (16-core) system, you must also have the associated module, xtpe-barcelona, xtpe-shanghai, xtpe-istanbul, xtpe-mc8, xtpe-mc12, or xtpe-interlagos loaded when compiling and linking your code. If one of these modules is loaded, the default target_system changes to the corresponding cpu target.

If the target_system is set during compilation of any source file, it must also be set to that same target during linking and loading.

FORMAT_TYPE_CHECKING

Specifies various levels of conformance between the data type of each I/O list item and the formatted data edit descriptor.

When set to RELAXED, the run-time I/O library enforces limited conformance between the data type of each I/O list item and the formatted data edit descriptor.

When set to STRICT77, the run-time I/O library enforces strict FORTRAN 77 conformance between the data type of each I/O list item and the formatted data edit descriptor.

When set to STRICT90 or STRICT95, the run-time I/O library enforces strict Fortran 90/95 conformance between the data type of each I/O list item and the formatted data edit descriptor.

FORTRAN_MODULE_PATH

As with the ftn -p module_site command line option, this environment variable enables you to specify the files or directory to search for the modules to use. The files can be archive files, build files (bld files), or binary files.

The compiler appends the paths specified by the FORTRAN_MODULE_PATH environment variable to path specified by the -p module_site command line option.

Since the FORTRAN_MODULE_PATH environment variable can specify multiple files and directories, a colon separates each path as shown in the following example:

```
% set FORTRAN_MODULE_PATH='path1 : path2 : path3'
```

LISTIO_PRECISION

The LISTIO_PRECISION environment variable controls the number of digits of precision printed by list-directed output. The LISTIO_PRECISION environment variable can be set to FULL or PRECISION.

¬∑ FULL prints full precision (default).

¬∑ PRECISION prints x or x + 1 decimal digits, where x is value of the PRECISION intrinsic function for a given real value. This is a smaller number of digits, which usually ensures that the last decimal digit is accurate to within 1 unit. This number of digits is usually insufficient to assure that subsequent input will restore a bit-identical floating point value.

NLSPATH   Specifies the message system library catalog path. This environment variable affects compiler interactions with the message system. For more information on this environment variable, see the catopen(3c) man page.

NPROC   Specifies the maximum number of processes to be run. Setting NPROC to a number other than 1 can speed up a compilation if machine resources permit.

The effect of NPROC is seen at compilation time, not at execution time. NPROC requests a number of compilations to be done in parallel. It affects all the compilers and also make.

For example, assume that NPROC is set as follows:

```
setenv NPROC 2
```

The following command is entered:

```
ftn -o t main.f sub.f
```

In this example, the compilations from .f files to .o files for main.f and sub.f happen in parallel, and when both are done, the load step is performed. If NPROC is unset, or set to 1, main.f is compiled to main.o; sub.f is compiled to sub.o, and then the link step is performed.

You can set NPROC to any value, but large values can overload the system. For debugging purposes, NPROC should be set to 1. By default, NPROC is 1.

TMPDIR
: Specifies the directory containing the temporary files. The location of the directory is defined by your administrator and cannot be changed.

ZERO_WIDTH_PRECISION
: The ZERO_WIDTH_PRECISION environment variable controls the field width when field width w of Fw.d is zero on output. The ZERO_WIDTH_PRECISION environment variable can be set to PRECISION or HALF.

¬∑ PRECISION specifies that full precision will be written. This is the default.

¬∑ HALF specifies that half of the full precision will be written.

Cray Fortran Compiler recognizes these run time environment variables (for other run time environment variables, see Cray Fortran Reference Manual):

CRAY_MALLOPT_OFF
: If set, then the system default mallopt parameters are used, instead of the compiler default parameters. For most programs, run time performance is improved by using the compiler defaults, but more memory may be used.

MALLOC_MMAP_MAX_
: Specifies the maximum number of memory chunks to allocate with mmap. The compiler default value is 0. For most programs, run time performance is improved by using the compiler default, but more memory may be used.

MALLOC_TRIM_THRESHOLD_
: Specifies the minimum size of the unused memory region at the top of the heap before the region is returned to the operating system. The compiler default value is 536870912 bytes. For most programs, run time performance is improved by using the compiler default, but more memory may be used.

NO_STOP_MESSAGE

If set, and if the STOP [stop_code] statement does not specify the optional stop_code, then STOP messages are not produced when this statement is executed.

FILES

Files containing Fortran source code have names with one of the following extensions: .f, .F, .f90, .F90, .f95, .F95, .f03, .F03, .f08, .F08, .ftn, or .FTN. By default, several files are created during processing. The Cray Fortran Compiler adds a suffix to the file portion of the file name and places the files it creates into your working directory.

The loader produces an executable file (by default a.out). See the -o out_file option for information about specifying a different file name for the executable. If only one source file is specified on the command line, the .o file is created and deleted. To retain the .o file, use the -c option.

The following files are accepted or produced by the compiler:

a.out	Default name of the executable output file.
file.a	Library file to be searched for external references.
file.f file.F	Input Fortran source file in fixed source form. If file ends in .F, the source preprocessor is invoked.
file.f90 file.F90 file.f95 file.F95 file.f03 file.F03 file.f08 file.F08 file.ftn file.FTN	Input Fortran source file in free source form. If the file extension is .F90, .F95, .F03, .F08, or .FTN, the source preprocessor is invoked.
file.i	File containing output from the source preprocessor.
file.lst	Listing file.
file.o	Relocatable object file.

```
     file.s    Assembly language file.

     modulename.mod
               If the -em option is specified, the compiler writes a
               modulename.mod file for each module; modulename is created
               by taking the name of the module and, if necessary,
               converting it to uppercase. This file contains module
               information, including any contained procedures.

SEE ALSO
      as(1), ftnlx(1), explain(1), intro_directives(1), make(1),

     Cray Fortran Reference Manual
```