NEC

SX-Aurora TSUBASA

SX-Aurora TSUBASA

C/C++ Compiler User's Guide

Proprietary Notice

Remarks:

- This document is the revision 28th issued in Aug 2022.
- NEC C/C++ Compiler conforms to the following language standards.
  - ISO/IEC 9899:2011 Programming languages - C
  - ISO/IEC 14882:2014 Programming languages - C++
  - ISO/IEC 14882:2017 Programming languages - C++
  - OpenMP Application Program Interface Version 4.5
- NEC C/C++ Compiler also conforms a part of “ISO/IEC 14882:2020 Programming languages - C++” and ”OpenMP Application Program Interface Version 5.0”
- In this document, the Vector Engine is abbreviated as VE.
- The reader of this document assumes that you have knowledge of software development in Fortran/C/C++ language on Linux.
- All product, brand, or trade names in this publication are the trademarks or registered trademarks of their respective owners.
- The Apache License version 2.0 with LLVM Exceptions product is included by this product.

Contents

Chapter1 C/C++ Compiler

1.1 Overview

The NEC C/C++ compiler is a compiler that compiles and links C/C++ programs and creates binaries for execution on the CPU of the VE. This compiler implements the following optimization function so that VE hardware performance can be easily drawn to the limit.

- Vectorization
- Automatic Parallelization and OpenMP Parallelization
- Automatic Inlining
- Performance Information collection

With various compiler options, you can use these capabilities to the utmost while selecting these functions. For details of the optimization function and compiler options, refer to Chapter 2 and later.

1.2 Usage of the Compiler

(1) Setting Environment Variables

If you want to omit the path specification when starting the NEC C/C++ compiler, set the path to the environment variable **PATH**. The NEC C/C++ compiler is installed by default under /opt/nec/ve. Add /opt/nec/ve/bin to the environment variable **PATH**.

Although the NEC C/C++ compiler provides environment variables for setting paths such as header files and libraries, the NEC C/C++ compiler automatically searches for the default path, so you can use it without setting these environment variable. Set environment variables when you need to search nonstandard directories, such as when you always want to add OSS header files and library paths not included in the compiler.

For the environment variables, see “2.2 Environment Variables Referenced During Execution”.

(2) Examples

The following shows examples of invoking the C/C++ compiler. See “Chapter3 Compiler Options” for details of the compiler options.

- Compiling and linking a C source file (a.c).

```
$ ncc a.c
```

- Compiling and linking more than one source file.

```
$ ncc a.c b.c
```

- Compiling, linking, and naming an executable file.

```
$ ncc -o prog.out a.c
```

- Compiling and linking with the highest vectorization and optimization.

```
$ ncc -O4 a.c
```

- Compiling and linking with safe vectorization and optimization.

```
$ ncc -O1 a.c
```

- Compiling and linking without vectorization and optimization.

```
$ ncc -O0 a.c
```

- Compiling and linking using automatic parallelization.

```
$ ncc -mparallel a.c
```

- Compiling and linking using automatic inlining.

```
$ ncc -finline-functions a.c
```

- Compiling and linking using a compiler of specific version.

```
$ /opt/nec/ve/bin/ncc-X.X.X a.c (X.X.X is version of a compiler.)
```

1.3 Execution

The example when executing a program below.

- Executing a compiled program

```
$ ./a.out
```

- Executing with number of VE

```
$ env VE_NODE_NUMBER=1 ./a.out              (Execute on number 1 of VE)
```

- Executing with input file and input parameter.

```
$ ./a.out data1.in 10      (input the file ” data.in”  and value ” 10” )
```

- Executing with redirecting an input file.

```
$ ./a.out < data2.in
```

- Executing a parallelized program with specifying the number of threads.

```
$ ncc -mparallel -O3 a.c b.c
$ export OMP_NUM_THREADS=4
$ ./a.out
```

- Using the profiler (ngprof).

  The performance information file gmon.out is output at execution a program which compiled with **-pg** at compiling and linking. The contents of gmon.out can be analyzed and output using the command ngprof.

```
$ ncc -pg a.c
$ ./a.out
$ ls gmon.out
gmon.out
$ ngprof
(The performance information is output.)
```

1.4 Command Line Syntax

The command line syntax of invoking the compiler is as follows.

```
ncc [ compiler-option | file ] ...
nc++ [ compiler-option | file ] ...
```

1.5 Specifying Compiler Options

- The compiler option must begin with a hyphen "-". In addition, there must be a blank between compiler options.

Example:

```
$ ncc -v -c a.c          (Correct)
$ ncc -vc a.c                (Incorrect)
```

- The C/C++ Compiler recognizes the input file suffixes as follows. The other file suffixes are treated as an object file.

Suffix	Recognized File
.c .i	C source file
.h	C header file
.C .cc .cpp .cp .cxx .c++ .ii	C++ source file
.H .hh .hpp .hp .hxx .h++ .tcc	C++ header file
.S .s	Assembler source file

- The compiler options and input files can be specified using option files.
  An option file is used to specify compiler options that are always enabled at the invoking of the C/C++ Compiler. Compiler options can be specified in the same way as when the command line is used. The option files must be placed in the home directory, to which the environment variable **HOME** has been set.

Compiler Type	Option File Name
ncc	$HOME/.nccinit
nc++	$HOME/.nc++init

Example:

```
$ cat ~/.nccinit
-O3 -finline-functions
$ ncc -v a.c
/opt/nec/ve/libexec/ccom … -O3 -finline-functions … a.c
```

1.6 Searching files specified by #include directive

The C/C++ compiler searches the following directories in the following order for header files included by #include <*file-name*>.

Note The compiler also searches the directory where source file exists to find the files included by #include "*file-name*". The directory is searched at first.

(a) Directories specified by **-I**

(b) Subdirectory named “include” under the directory specified by **-B**
(c) Directories specified by the environment variable **NCC_INCLUDE_PATH**
(d) Directory specified by **-isystem**
(e) /opt/nec/ve/ncc/<*version-number*>/include
(f) Subdirectory named “include” under the directory specified by **-isysroot** if it is specified, otherwise /opt/nec/ve/include

1.7 Searching Libraries

The compiler searches the following directories in the following order for libraries.

(a) Directories specified by **-L**
(b) Directories specified by **-B**
(c) Directories specified by the environment variable **NCC_LIBRARY_PATH**
(d) /opt/nec/ve/ncc/<*version-number*>/lib
(e) Directories specified by the environment variable **VE_LIBRARY_PATH**
(f) /opt/nec/ve/lib/gcc
(g) /opt/nec/ve/lib

1.8 Arithmetic Exceptions

1.8.1 Operation Result After Arithmetic Exception Occurrence

This section describes how an overflow, underflow, division by zero, invalid operation, and accuracy degradation are handled when they occur during an arithmetic operation.

(1) Division by zero

When a division by zero occurs during an integer arithmetic operation, the result is undefined. When a division by zero occurs during a non-integer arithmetic operation, the result of the operation is the maximum expressible value if the dividend is positive, or the minimum expressible value if the dividend is negative. When the value of **VE_FPE_ENABLE** is “DIV”, this exception occurs and error message is issued to the standard error output. When the value of **VE_FPE_ENABLE** is not “DIV”, this exception does not occurs.

(2) Floating-point overflow

When an overflow occurs during an operation of type real and complex, the result

of the operation is the maximum expressible value if the value is positive, or the minimum expressible value if the value is negative.
When the value of **VE_FPE_ENABLE** is “FOF”, this exception occurs and error message is issued to the standard error output. When the value of **VE_FPE_ENABLE** is not “FOF”, this exception does not occurs.

(3) Floating-point underflow
When an underflow occurs during an operation of type real and complex, the result of the operation is zero.
When the value of **VE_FPE_ENABLE** is “FUF”, this exception occurs and error message is issued to the standard error output. When the value of **VE_FPE_ENABLE** is not “FUF”, this exception does not occurs.

(4) Invalid operation
When an invalid operation occurs during an operation of type real and complex, the result of the operation is an undefined value or **NaN**.
When the value of **VE_FPE_ENABLE** is “INV”, this exception occurs and error message is issued to the standard error output. When the value of **VE_FPE_ENABLE** is not “INV”, this exception does not occurs.

(5) Accuracy degradation
When accuracy degradation occurs during an operation of type real and complex, the result of the operation is a rounded value.
When the value of **VE_FPE_ENABLE** is “INE”, this exception occurs and error message is issued to the standard error output. When the value of **VE_FPE_ENABLE** is not “INE”, this exception does not occurs.

(6) Exception while executing a vector instruction
When overflow, underflow, or division by zero occurs while executing a vector instruction, the processing is the same as in the case of a scalar instruction. However, if multiple operation exceptions occur at the same time while executing one vector instruction, they appear as one exception.

1.8.2 Changing Arithmetic Exception Mask

By changing the mask setting, it can be specified whether an arithmetic exception occurs or not.
The arithmetic exception mask can be changed by using **VE_FPE_ENABLE**. Which

kind of mask should be changed must be specified by **VE_FPE_ENABLE**.

Example:

```
$ export VE_FPE_ENABLE=FOF,DIV
$ ./a.out
```

In the above example, changing the mask setting so that Floating-point overflow (FOF) or Divide-by-zero exception (DIV) can occur.

1.8.3 Using Traceback Information

Where the arithmetic exception occurred can be ascertained by changing the mask and using the traceback information.

Example:

```
$ ncc -traceback=verbose below.c out.c watch.c hey.c ovf.c
...
$ export VE_TRACEBACK=VERBOSE
$ export VE_FPE_ENABLE=DIV
$ ./a.out
Runtime Error: Divide by zero at 0x600008001088
[ 0] 0x600008001088 below_           below.c:3
[ 1] 0x600018001168 out_             out.c:3
[ 2] 0x600020001168 watch_           watch.c:3
[ 3] 0x600010001168 hey_             hey.c:3
[ 4] 0x60000001cab8 MAIN__           ovf.c:5
```

In example, the exception of “Divide by zero” occurred in line 3 of below.c.

1.8.4 Remarks on Changing Arithmetic Exception Mask

Changing the arithmetic exception mask affects the system library functions called from a program. Therefore, the arithmetic exception is raised if precision degradation or another exception occurs in the system library functions.

Chapter2 Environment Variables

2.1 Environment Variables Referenced During Compilation

HOME

This variable is referenced by the compiler in order to search the user's home directory for an option file. When **HOME** is not set, the option file has no effect even if it is put on the home directory.

NCC_COMPILER_PATH

Specified a list of directories separated by colon which are searched for the C/C++ compiler (ccom). The directory has high priority in the order of listing. If it is not found in the specified directories, ncc/nc++ starts the C/C++ compiler in the standard directory. This environment variable is set when you want to always search non-standard directories.

Example:

```
$ export NCC_COMPILER_PATH= "$HOME/libexec:$HOME/wk/libexec"
```

NCC_INCLUDE_PATH

Specifies a list of directories separated by colon which are searched for the header files. The directory has high priority in the order of listing. This environment variable is set when you want to always search non-standard directories. For example, you want to always search the OSS header file directory that is not attached to the NEC C/C++ compiler.

Example:

```
$ export NCC_INCLUDE_PATH= "$HOME/include:$HOME/wk/include"
```

NCC_LIBRARY_PATH

Specifies a list of directories separated by colon which are searched for the C/C++ libraries. The directory has high priority in the order of listing. This environment variable is set when you want to always search non-standard directories. For example, you want to always search the OSS library directory that is not attached to the NEC C/C++ compiler.

Example:

```
$ export NCC_LIBRARY_PATH= “$HOME/lib:$HOME/wk/lib”
```

NCC_PROGRAM_PATH

Specified a list of directories separated by colon which are searched for the assembler and the linker for VE. The directory has high priority in the order of listing. If they are not found in the specified directories, the NEC C/C++ compiler automatically starts the assembler and linker in the standard directory. This environment variable is set when you want to always search non-standard directories.

Example:

```
$ export NCC_PROGRAM_PATH= “$HOME/bin:$HOME/wk/bin”
```

PATH

Add a list of directories separated by colon which are searched for the ncc/nc++. The directory has high priority in the order of listing. Add the "bin" under the directory where the NEC C/C++ compiler is installed. If you set this environment variable, you can omit specifying the path when starting ncc/nc++. When installing to the standard directory, add "/opt/nec/ve/bin". The environment variable **PATH** also affects other applications of the NEC C/C++ compiler. Add it to the existing environment variable **PATH**.

Example:

```
$ export PATH= “/opt/nec/ve/bin:$PATH”
```

TMPDIR

Specifies a directory where the compilers and commands temporarily use. (default: /tmp)

VE_LIBRARY_PATH

Specifies a list of directories separated by colon which are searched for the system libraries. The directory has high priority in the order of listing. This environment variable is set when you want to always search non-standard directories.

Example:

```
$ export VE_LIBRARY_PATH= “$HOME/lib:$HOME/wk/lib”
```

2.2 Environment Variables Referenced During Execution

OMP_NUM_THREADS / VE_OMP_NUM_THREADS

This variable sets the number of threads to use for OpenMP and/or automatic parallelized programs. The number of threads is the number of cores of the VE when it is not specified explicitly.

Example:

```
$ export OMP_NUM_THREADS=4
```

OMP_STACKSIZE / VE_OMP_STACKSIZE

This variable sets the upper limit of the stack size by the kilobytes used by each threads for OpenMP and/or automatic parallelized programs. The value can be specified as megabytes by using M as unit and gigabytes by using G as unit.

Example:

```
$ export OMP_STACKSIZE=1G
```

OMP_TOOL / VE_OMP_TOOL

This variable is used to enable or disable OMPT interface. When “enabled” is set, OMPT interface is enabled. In default, it is disabled.

Example:

```
$ export OMP_TOOL=enabled
```

OMP_TOOL_LIBRARIES / VE_OMP_TOOL_LIBRARIES

This variable is used to set a dynamic-loaded library for OMPT interface. Specify colon (:) to specify two or more libraries.

Example:

```
$ export OMP_TOOL_LIBRARIES=libomptool.so:/usr/myhome/libompt.so
```

VE_ADVANCEOFF

This variable is used to control the advance-off (lockstep execution) mode. When

"YES" is set, the advance-off mode is enabled.

If any other value is set or this variable is not set, the advance-off mode is disabled.

If the advance-off mode is enabled, the execution time can be significantly increased.

Example:

```
$ export VE_ADVANCEOFF=YES
```

VE_FPE_ENABLE

This variable is used to control over floating-point exception handling at run-time. When this variable is set, then the specified exception is enabled.

The value of this variable is a comma separated list, each element of which is one of the following values.

DIV

Divide-by-zero exception.

FOF

Floating-point overflow exception.

FUF

Floating-point underflow exception.

INV

Invalid operation exception.

INE

Inexact exception.

Example:

```
$ export VE_FPE_ENABLE=DIV
```

VE_INIT_STACK

This variable sets the value to initialize the stack area at the run-time. When the value is not set, the stack area is initialized with zeros. **-minit-stack=runtime** is needed at compilation. The following values can be specified.

ZERO

Initializes with zeros.

NAN

Initializes with quiet NaN in double type (0x7fffffff7fffffff).

NANF

Initializes with quiet NaN in float type (0x7fffffff).

SNAN

Initializes with signaling NaN in double type (0x7ff4000000000000).

SNANF

Initializes with signaling NaN in float type (0x7fa00000).

0x*XXXX*

Initializes with the value specified in a hexadecimal format up to 16 digits. When the specified value has more than 8 hexadecimal digits, the initialization is done on an 8-byte cycle. Otherwise it is done on a 4-byte cycle.

Example:

```
$ ncc -minit-stack=runtime a.c
$ export VE_INIT_STACK=SNAN
$ ./a.out
```

VE_LD_LIBRARY_PATH

This variable set a list of directories separated by colon that the dynamic linker searches for libraries. The dynamic linker automatically searches the standard directories. This environment variable is set when you want to always search non-standard directories. For example, you want to always search the OSS library directory that is not attached to the NEC C/C++ compiler.

Example:

```
$ export VE_LD_LIBRARY_PATH= “${HOME}/lib:$VE_LD_LIBRARY_PATH”
```

VE_NODE_NUMBER

This variable is set to designate a program to be executed on specified VE node.

VE_PROGINF

When “YES” or “DETAIL” is set, the program execution information is output to the standard error output at the termination of execution. See the manual ”PROGINF/FTRACE User’s Guide” for the detail.

VE_TRACEBACK

This variable is used to control to output traceback information when a fatal error

occurs at runtime. The program must be compiled and linked with **-traceback** to output traceback information. When the value of this variable is “FULL” or “ALL”, then at most depth which is specified by **VE_TRACEBACK_DEPTH** environment variable of traceback information is output. If any other value is set, only traceback information of the function that a fatal error occurs is output. If this variable is not set, no traceback information is output.
An occurrence line number of fatal error is found by address information in traceback information.

Example:

```
$ export VE_TRACEBACK=FULL
$ ./a.out
Runtime Error: Divide by zero at 0x600000000cc0
[ 1] Called from 0x7f5ca0062f60
[ 2] Called from 0x600000000b70
Floating point exception
```

The line number can be sought from the address information using the command naddr2line. In example, the exception of “Divide by zero” occurred in line 3 of a.c.

Example:

```
$ naddr2line -e ./a.out -a 0x600000000cc0
0x0000600000000cc0
/.../a.c:3
```

When running the program which is compiled and linked with **-traceback=verbose** and the value of this variable is “VERBOSE”, filename and line number is output in traceback information.

Example:

```
$ export VE_TRACEBACK=VERBOSE
$ ./a.out
Segmentation fault: Address not mapped to object at 0x600008001078
[ 0] 0x600008001078 below            below.c:8
[ 1] 0x600018001170 out              out.c:3
[ 2] 0x600020001170 watch            watch.c:3
[ 3] 0x600010001170 hey              hey.c:3
[ 4] 0x600000001500 main             ovf.c:10
```

VE_TRACEBACK_DEPTH

This variable is used to control the maximum depth of traceback information when

it is output. When it is not specified explicitly, then 50 is set. If 0 is specified, then the maximum depth is unlimited.

Chapter3 Compiler Options

This chapter describes the operating procedures for compiling, linking, and executing a C/C++ program using the C/C++ compiler system.
The compiler options of the C/C++ compiler can be divided into the following categories.

- Overall Options
  Compiler options used to control the C/C++ compiler.
- Optimization Options
  Compiler options used to control optimization and vectorization.
- Parallelization Options
  Compiler options used to control parallelization.
- Inlining Options
  Compiler options used to control inlining.
- Code Generation Options
  Compiler options used to control code generation for performance measurement and the stack area initialization.
- Debug Options
  Compiler options used to control debug code generation.
- Language Options
  Compiler options used to enable or disable language features.
- Message Options
  Compiler options used to control message output.
- List Output Options
  Compiler options used to control compiler listing.
- Preprocessor Options
  Compiler options used to control preprocessing.
- Assembler Options
  Compiler options used to specify assembler functions.
- Linker Options
  Compiler options used to specify linker functions.

- Directory Options

  Compiler options used to specify various directories.

3.1 Overall Options

-S

Suppresses the linking and outputs the assembler source file.

-c

Suppresses the linking and outputs the object file.

-cf=*conf*

Applies the configuration file specified by *conf* to compilation and linking.

-clear

Ignores all compiler options and input files specified before **-clear**.

-fsyntax-only

Performs only grammar analysis.

-o *filename*

Specifies a *filename* to which output is written, where the output is preprocessed text, assembler source file, object file or executable file. This option cannot be specified when two or more source files are specified with **-S**, **-c**, or **–E**.

-x *language*

Specifies the *language* kind for the input files. The effect of this option is prior to the default setting according to the file suffix and the specification is applied to all the input files following this option (until the next **-x** if any) on the command-line. One of the following can be specified as language.

c

Compiles as a C source file.

c++

Compiles as a C++ source file.

assembler

Assembles as an assembler source file.

assembler-with-cpp

Does preprocessing and assembles the preprocessed file.

@*file-name*

Reads options from *file-name* and inserts them in the place of the original **@*file-name*** option.

3.2 Optimization Options

-O[*n*]

Specifies optimization level by *n*. The following are available as *n*:

4

Enables aggressive optimization which violates language standard.

3

Enables optimization which causes side-effects and nested loop optimization.

2

Enables optimization which causes side-effects. (default)

1

Enables optimization which does not cause any side effects.

0

Disables any optimizations, automatic vectorization, parallelization, and inlining.

-fargument-alias

Allows the compiler to assume that arguments are aliasing each other and non-local-objects in all optimization. (default)

-fargument-noalias

Disallows the compiler to assume that arguments are aliasing each other and non-local-objects in all optimization.

-f[no-]associative-math

Allow [Disallows] re-association of operands in series during optimization and loop transformation. When **-fno-associative-math** is specified, the optimization which transform matrix multiply loops into a vector matrix library function call with **-fmatrix-multiply** is not performed. (default: **-fassociative-math**)

-f[no-]aggressive-associative-math

Allows [Disallow] aggressive re-association of operands in series during optimization and loop transformation.
(default: **-fno-aggressive-associative-math**)

-f[no-]check-noexcept-violation

Enable [Disable] runtime checking weather C++ noexcept specification is violated or not. When this option is not effective, std::terminate function is not called and a program execution continues even if noexcept specification is violated.
(default: **-fno-check-noexcept-violation**)

-f[no-]cse-after-vectorization

[Does not] Re-apply common subexpression elimination after vectorization. (default: **-fno-cse-after-vectorization**)

-f[no-]fast-math

[Does not] uses fast scalar version math functions outside of vectorized loops. (default: **-ffast-math**)

-f[no-]ignore-induction-variable-overflow

[Does not] Ignores induction variable overflow in optimization. (default: **-fno-ignore-induction-variable-overflow**)

-f[no-]ignore-volatile

[Does not] Ignores volatile attribute in optimization. (default: **-fno-ignore-volatile**)

-fivdep

Inserts **ivdep** directive before all loops.

-fivdep-omp-worksharing-loop

Inserts **ivdep** directive before an OpenMP parallelized loop that does not have **simd** with **safelen** and/or **simdlen** clause.

-f[no-]loop-collapse

Allows [Disallows] loop collapsing. **-O***n* (*n*=2,3,4) must be effective. (default: **-fno-loop-collapse**)

-floop-count=*n*

Specifies *n* which is taken to assume the iteration count of the loop whose iteration count cannot be decided at compilation to do optimization suitable for loop count. (default: **-floop-count**=5000)

-f[no-]loop-fusion

Allows [Disallows] loop fusion. **-O***n* (*n*=2,3,4) must be effective. (default: **-fno-loop-fusion**)

-f[no-]loop-interchange

Allows [Disallows] loop interchange. **-O***n* (*n*=2,3,4) must be effective. (default: **-fno-loop-interchange**)

-f[no-]loop-normalize

Allows [Disallows] loop normalization. Compiler assumes that loop iteration count is not changed in loop body. (default: **-fno-loop-normalize**)

-f[no-]loop-split

Allows [Disallows] splitting out of a function call in a loop from the loop. **-O***n* (*n*=2,3,4) must be effective. (default: **-fno-loop-split**)

-f[no-]loop-strip-mine

Allows [Disallows] loop strip mining. **-O***n* (*n*=2,3,4) must be effective. (default: **-fno-loop-strip-mine**)

-f[no-]loop-unroll

Allows [Disallows] loop unrolling. **-O***n* (*n*=2,3,4) must be effective. (default: **-floop-unroll**)

-floop-unroll-complete=*m*

Allows loop expansion (complete loop unrolling) of a loop whose iteration count is constant, can be calculated, and is less than or equal to *m*. **-O***n* (*n*=2,3,4) must be effective. (default: **-floop-unroll-complete**=4)

Remark:

-floop-unroll-completely=*m* can be used as an alias option name.

-floop-unroll-max-times=*n*

Specifies maximum unrolled times by *n*. When this option is not effective, the compiler automatically choose the suitable unroll times.

-f[no-]matrix-multiply

Allows [Disallows] to transform matrix multiply loops into a vector matrix library function call. **-O***n* (*n*=2,3,4) and **-fassociative-math** must be effective. (default: **-fno-matrix-multiply**)

-f[no-]move-loop-invariants

Enable [Disables] the loop invariant motion under if-condition. (default: **-fmove-loop-invariants**)

-f[no-]move-loop-invariants-if

Allows [Disallows] the loop invariant if-structure motion. **-O***n* (*n*=2,3,4) must be effective. (default: **-fno-move-loop-invariants-if**)

-f[no-]move-loop-invariants-unsafe

Allow [Disallow] motion of unsafe codes which may cause any side effects.

The example of unsafe codes are:

- divide
- memory reference to 1 byte or 2 byte area

(default: **-fno-move-loop-invariants-unsafe**)

-f[no-]move-nested-loop-invariants-outer

Allows [Disallows] the compiler to move the loop invariant expressions to outer loop. When this option is specified, they are moved before the current loop. (default: **-fmove-nested-loop-invariants-outer**).

-fnaked-ivdep

Accepts "**#pragma ivdep**" as "**#pragma _NEC ivdep**".

-fnamed-alias

The compiler will assume that the object pointed-to-by a named pointer have an alias in applying optimization and vectorization.

-fnamed-noalias

The compiler will assume that the object pointed-to-by a named pointer does not have an alias in applying optimization and vectorization. (default)

-f[no-]outerloop-unroll

Allows [Disallows] outer-loop unrolling. **-O***n* (*n*=2,3,4) must be effective. (default: **-fno-outerloop-unroll**)

-fouterloop-unroll-max-size=*n*

Specifies maximum size of an innermost loop to be outer-loop-unrolled. (default: **-fouterloop-unroll-max-size**=4)

-fouterloop-unroll-max-times=*n*

Specifies maximum outer-loop unrolled times by *n*. *n* must be power of 2. When this option is not effective, the compiler automatically choose the suitable unroll times.

-f[no-]precise-math

[Does not] Apply high resolution algorithm in the vector versions of pow(3C) and powf(3C) when the exponent is an integer value. Their results become more exact but their calculation speeds become slower. (default: **-fno-precise-math**)

-f[no-]reciprocal-math

Allows [Disallows] change an expression "x/y" to "x * (1/y)". (default: **-freciprocal-math**)

-f[no-]replace-loop-equation

Allows [Disallows] replacement of "!=" and "==" operator with "<=" or ">=" at the loop backedge. (default: **-fno-replace-loop-equation**)

-f[no-]strict-aliasing

Allows [Disallows] the compiler to assume the ANSI aliasing rules in all optimization.

(default: **-fstrict-aliasing**)

When this option is not effective, the compiler assumes the stored value is accessed only by one of the following types.

- A type compatible with the effective type of the object
- A qualified version of a type compatible with the effective type of the object
- A type that is the signed or unsigned type corresponding to the effective type of the object
- A type that is the signed or unsigned type corresponding to a qualified version of the effective type of the object
- An aggregate or union type that includes one of the aforementioned types among its members (including, recursively, a member of a sub aggregate or contained union)
- A character type

-fthis-pointer-alias

Allows the compiler to assume that this-pointer has an alias in all optimization.

-fthis-pointer-noalias

Disallows the compiler to assume that this-pointer has an alias in all optimization. (default)

-m[no-]list-vector

Allows [Disallows] the vectorization of the statement in a loop when an array element with a vector subscript expression appears on both the left and right sides of an assignment operator. (default: **-mno-list-vector**)

-mretain-*keyword*

Sets higher priority to vector memory access results to retain on LLC (Last-Level Cache). The following are available as *keyword*:

all

Sets higher priority to vector load/store/gather/scatter results. (default)

list-vector

Sets higher priority to vector gather/scatter results.

none

Does not set higher priority to vector memory access results.

-msched-*keyword*

Specifies whether and how the instruction scheduling. The following are available

as *keyword*.

none

Does not perform the instruction scheduling.

insns

Performs the instruction scheduling in a basic block.

block

Performs the instruction scheduling in a basic block, but to a wider range than **-msched-insns** does, in order to schedule instructions aggressively. (default)

interblock

Performs the instruction scheduling beyond basic blocks.

-m[no-]vector

Enable [Disables] automatic vectorization. (default: **-mvector**)

-m[no-]vector-advance-gather

Allows [Disallows] motion of vector gather instructions so that they can be started as advance as possible. (default: **-mvector-advance-gather**)

-mvector-advance-gather-limit=*n*

The number of vector gather operations which is moved by **-mvector-advance-gather** is up to *n*. (default: **-mvector-advance-gather-limit**=56)

-m[no-]vector-dependency-test

Allows [Disallows] the conditional vectorization by dependency-test. **-O***n* (*n*=2,3,4) must be effective. (default: **-mvector-dependency-test**)

-m[no-]vector-floating-divide-instruction

Allows [Disallows] to use vector-floating-divide instruction. By default, approximate instruction sequence by using vector-floating-reciprocal instructions is used.

(default: **-mno-vector-floating-divide-instruction**)

-m[no-]vector-fma

Allows [Disallows] to use vector fused-multiply-add instruction.

(default: **-mvector-fma**)

-m[no-]vector-intrinsic-check

[Does not] Checks the value ranges of arguments in the mathematical functions the vectorized version. (default: **-mno-vector-intrinsic-check**)

The target mathematical functions of this option are as follows.

acos, acosh, asin, atan, atan2, atanh, cos, cosh, cotan, exp, exp10, exp2,

expm1, log10, log2, log, pow, sin, sinh, sqrt, tan, tanh

-m[no-]vector-iteration

Allows [Disallows] to use vector iteration instruction in the vectorization. (default: **-mvector-iteration**)

-m[no-]vector-iteration-unsafe

Allows [Disallows] to use vector iteration instruction in the vectorization when it may give incorrect result. (default: **-mvector-iteration-unsafe**)

-m[no-]vector-loop-count-test

Allows [Disallows] the conditional vectorization by loop-iteration-count-test. **-O***n* (*n*=2,3,4) must be effective. (default: **-mno-vector-loop-count-test**)

-m[no-]vector-low-precise-divide-function

Allows [Disallows] to use low precise version for vector floating divide operation. It is faster than the normal precise version but the result may include at most one bit numerical error in mantissa. (default: **-mno-vector-low-precise-divide-function**)

-m[no-]vector-merge-conditional

Allows [Disallows] to merge vector load and store in **THEN** block, **ELSE IF** block, and **ELSE** block. (default: **-mno-vector-merge-conditional**)

-m[no-]vector-packed

Allows [Disallows] to use packed vector instruction. (default: **-mno-vector-packed**)

-m[no-]vector-power-to-explog

Allows [Disallows] to replace pow(R1,R2) in a vectorized loop with exp(R2*log(R1)). powf(3C) is replaced, too. By the replacement, the execution time would be shortened, but numerical error occurs rarely in the calculation. (default: **-mno-vector-power-to-explog**)

-m[no-]vector-power-to-sqrt

Allows [Disallows] to replace pow(R1,R2) in a vectorized loop with the expression including sqrt(3C) or cbrt(3C) when R2 is a special value such as 0.5, 1.0/3.0 etc. powf(3C) is not replaced, too. When it is replaced, the execution time would become faster, but numerical error occurs rarely in the calculation. (default: -mvector-p**ower-to-sqrt**)

-m[no-]vector-reduction

Allows [Disallows] to use vector reduction instruction in the vectorization.

(default: **-mvector-reduction**)

-m[no-]vector-shortloop-reduction

Allows [Disallows] the conditional vectorization by loop-iteration-test for reduction. **-O***n* (*n*=2,3,4) must be effective.

(default: **-mno-vector-shortloop-reduction**)

-m[no-]vector-sqrt-instruction

Allows [Disallows] to use vector-sqrt instruction. By default, approximate instruction sequence by using vector-floating-reciprocal instructions is used.

(default: **-mno-vector-sqrt-instruction**)

-mvector-threshold=*n*

Specifies the minimum iteration count (*n*) of a loop for vectorization.

(default: **-mvecter-threshold**=5)

-mwork-vector-kind=none

Disallows the partial vectorization using loop division.

3.3 Parallelization Options

-fopenmp

Enables OpenMP directives. **-pthread** is implicitly enabled.

-fopenmp-tools

Enables OMPT interface. (default: **-fno-openmp-tools**)

-m[no-]create-threads-at-startup

[Does not] Generates threads for OpenMP or automatic parallelization at the first parallel region execution. The threads are generated at the startup of the execution at default. (default: **-mcreate-threads-at-startup**)

Remark:

-static-nec or **-static** must be specified when you specified this option.

-mparallel

Allows automatic parallelization. **-pthread** is implicitly enabled.

-mparallel-innerloop

Allows to parallelize inner-loop.

-m[no-]parallel-omp-routine

Allows [Disallows] to apply automatic parallelization to a routine including OpenMP directive.

(default: **-mparallel-omp-routine**)

-mparallel-outerloop-strip-mine

Allows to parallelize the nested loops that are outer-loop strip-mined.

-mparallel-sections

Allows to generate parallelized sections.

-mparallel-threshold=*n*

Specifies the threshold value *n* of the loop parallelization. When the value is larger than the work of the loop, the loop is parallelized.

(default: **-mparallel-threshold**=2000)

-mschedule-dynamic

-mschedule-runtime

-mschedule-static

-mschedule-chunk-size=*n*

Specifies a scheduling kind and chunk size of a thread when they are not specified by schedule-clause in OpenMP parallelization and automatic parallelization.

-pthread

Enables support for multithreading with the pthread library.

3.4 Inlining Options

-finline-abort-at-error

Stops the compilation when generation of routines defined in source files fails. Does not search them and continues the compilation when this option is not effective.

(default: **-fno-inline-abort-at-error**)

-f[no-]inline

Allows [Disallows] the inlining of inline functions. (default: **-finline**)

-f[no-]inline-copy-arguments

[Does not] Generate a copy of the argument of an inlined function call by automatic inlining. When not generating a copy the function parameter is replaced with a corresponding function argument.

(default: **-finline-copy-arguments**)

-finline-directory=*directory name*

Searches all source files under directories separated by colon for functions to inline.

-fno-inline-directory=*directory name*

Does not search all source files under directories separated by colon for functions to inline. This option is specified when you do not want to search the source files specified by **-finline-file** or **-finline-directory**.

-finline-file=*string*

Searches source files separated by colon for functions to inline. Searches all input source files specified in command line when **all** is specified.

-fno-inline-file=*string*

Does not search source files separated by colon for functions to inline. This option is specified when you do not want to search the source files specified by **-finline-file** or **-finline-directory**.

-finline-functions

Allows automatic inlining.

-finline-max-depth=*n*

Specifies the level of functions to be inlined from the bottom of the calling tree by automatic inlining. (default: **-finline-max-depth**=2)

-finline-max-function-size=*n*

Specifies the function size (= the amount of intermediate representations for a function) to be inlined by automatic inlining.

(default: **-finline-max-function-size**=50)

-finline-max-times=*n*

Sets the limit of the function size (= the amount of intermediate representations for a function) after automatic inlining to “(function-size-before-inlining) * *n*”.

(default: **-finline-max-times**=6)

-f[no-]inline-suppress-diagnostics

[Does not] Output diagnostics when generation of routines defined in source files to search fails. The option **-fno-inline-suppress-diagnostics** is specified when you want to check which source files you specified are searched normally.

(default: **-finline-suppress-diagnostics**)

-mgenerate-il-file

Outputs an IL file for cross-file inlining. The file is created in the current directory, under the name "*source-file-name*.cil".

-mread-il-file *IL file name*

Read IL files separated by colon for functions to inline. When **-finline-directory**, **-finline-file** or **-mgenerate-il-file** are specified, this option is ignored.

3.5 Code Generation Options

-finstrument-functions

Inserts function calls for the instrumentation to entry and exit of functions. The instrumented functions are;

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

-fpic

-fPIC

Generates position-independent code.

-ftrace

Creates an object file and the executable file for ftrace function.

(default: **-no-ftrace**)

-p

-pg

Creates an executable file for output profiler information (ngprof).

-[no-]proginf

Does not create an executable file for PROGINF function. (default: **-proginf**)

3.6 Debugging Options

-g

Generates debugging information in DWARF.

-minit-stack=*value*

Initializes the stack area with the specified value at the run-time. The following are available as *value*:

zero

Initializes with zeros.

nan

Initializes with quiet NaN in double type (0x7fffffff7fffffff).

nanf

Initializes with quiet NaN in float type (0x7fffffff).

snan

Initializes with signaling NaN in double type (0x7ff4000000000000).

snanf

Initializes with signaling NaN in float type (0x7fa00000).

runtime

Initializes with the value specified by the environment variable **VE_INIT_STACK**.

0x*XXXX*

Initializes with the value specified in a hexadecimal format up to 16 digits. When the specified value has more than 8 hexadecimal digits, the initialization is done on an 8-byte cycle. Otherwise it is done on a 4-byte cycle.

-traceback[=verbose]

Specifies to generate extra information in the object file and to link run-time library due to provide traceback information when a fatal error occurs and the environment variable **VE_TRACEBACK** is set at run-time.

When **verbose** is specified, generates filename and line number information in addition to the above due to provide these information in traceback output. Set the environment variable **VE_TRACEBACK=VERBOSE** to output these information at run-time.

3.7 Language Options

3.7.1 C Language Control Options

-fno-allow-keyword-macros

Disallows to define any keyword macros.

-fgnu89-inline

Performs inlining according to semantic rules of the GNU C89 specification.

-f[no-]restrict

[Does not] Treat **restrict** as C keyword. (default: **-frestrict**)

Remark:

The default is different in C and C++. In C++, the default is **-fno-restrict**.

-fsigned-char | **-funsigned-char**

Specifies whether to treat plain char type as signed or unsigned.

(default: **-fsigned-char**)

-std=*standard*

Specifies C Language standard. Available keywords as language are gnu89,

gnu99, gnu11, c99 or c11. (default: **-std**=gnu11)

-traditional

Preprocesses C source file according to the K&R C language specification. This option must be specified with **-E**.

-traditional-cpp

Preprocesses C source file according to the K&R C language specification.

-trigraphs

Enables recognition of trigraph sequences.

3.7.2 C++ Language Control Options

-fdefer-inline-template-instantiation

Do not instantiate an inline function template at the function call position and postpone it to appropriate timing. (default).

-fno-defer-inline-template-instantiation

Do instantiate an inline function template at the function call position. (default: **-fdefer-inline-template-instantiation**).

-f[no-]exceptions

Enables [Disables] C++ exception handling feature. (default: **-fexceptions**)

-fext-numeric-literals

Treats a constant expression with a suffix I, i, J or j as a complex constant. (default when **-std**=gnu++11, **-std**=gnu++14 or **-std**=gnu++17 is effective)

-fno-ext-numeric-literals

Treats a constant expression with a suffix I, i, J or j as a user-defined literal. (default when **-std**=c++11, c++14, c++17 or c++20 is effective)

-ffor-scope

The scope of variables declared in a for-init-statement is limited to the "for" loop itself. (default)

-fno-for-scope

The scope of variables declared in a for-init-statement is extended to the end of the enclosing scope. (default: **-ffor-scope**)

-fimplicit-include

Allows implicit inclusion of source files as a method of finding definitions of template entities to be instantiated.

-f[no-]restrict

[Does not] Treats **restrict** as C++ keyword. (default: **-fno-restrict**)

Remark:

The default is different in C and C++. In C, the default is **-frestrict**.

-f[no-]rtti

Enables [Disables] run-time-type-identification feature. (default: **-frtti**)

-ftemplate-depth=*n*

Specifies the maximum number of instantiations of a given template that may be in process of being instantiated at a given time. This is used to detect runaway recursive instantiations. The value *n* can be between 0 and 1024. If *n* is zero, there is no limit. (default: **-ftemplate-depth**=256)

-std=*standard*

Specifies C++ Language standard. Available keywords as language are gnu++11, gnu++14, gnu++17, gnu++20, c++11, c++14, or c++17 or c++20. (default: **-std**=gnu++14)

3.8 Message Options

-Wall

Outputs all syntax warning messages.

-Wcomment

Outputs a warning message for a comment-start sequence /* which appears in a /* */ comment.

-Werror

Treats all syntax warnings as fatal errors.

-Wno-div-by-zero

Suppresses a warning message for integer division by zero detected at the compilation.

-Wunknown-pragma

Outputs a warning message when the compiler encounters unknown **#pragma**.

-Wunused

Same as **-Wunused-variable**.

-Wunused-but-set-parameter

Outputs a warning message for any parameters which is set but not used.

-Wunused-but-set-variable

Outputs a warning message for any local variables which is set but not used.

-Wunsued-parameter

Outputs a warning message for any parameters which is not used.

-Wunused-value

Outputs a warning message for any expressions whose value is computed but not used.

-Wunused-variable

Outputs a warning message for any local variables or functions which is not used.

-fdiag-inline=*n*

Specifies automatic inlining diagnostics level by *n*. (0: No output, 1: Information, 2: Detail) (default: **-fdiag-inline**=1)

-fdiag-parallel=*n*

Specifies automatic parallelization diagnostics level by *n*. (0: No output, 1: Information, 2: Detail) (default: **-fdiag-parallel**=1)

-fdiag-vector=*n*

Specifies vector diagnostics level by *n*. (0: No output, 1: Information, 2: Detail) (default: **-fdiag-vector**=1)

-fdiag-system-header

Outputs optimization diagnostics for a function defined in a system header.

-pedantic

Outputs the warnings for using of language extension.

-pedantic-errors

Outputs the errors for using of language extension.

-w

Suppresses all warning messages.

3.9 List Output Options

-report-file=*filename*

Outputs the listing result to the specified file instead of the default one.

-report-append-mode

Opens the output file with “appending mode” instead of “overwriting mode”. This option cannot be used unless the **-report-file** option is specified.

-report-all

Outputs the code generation list, diagnostic list, format list, inline list, option list

and vector list.

-[no-]report-cg

[Does not] Outputs optimization list of code generation module.

(default: **-no-report-cg**)

-[no-]report-diagnostics

[Does not] Outputs diagnostic list. (default: **-no-report-diagnostics**)

-[no-]report-format

[Does not] Outputs format list. (default: **-no-report-format**)

-[no-]report-inline

[Does not] Outputs optimization list of inlining module. (default: **-no-report-inline**)

-[no-]report-option

[Does not] Outputs option list. (default: **-no-report-option**)

-[no-]report-system-header

[Does not] Outputs compiler listings for a function defined in a system header.

(default: **-no-report-system-header**)

-[no-]report-vector

[Does not] Outputs optimization list of vectorization module.

(default: -no**-report-vector**)

3.10 Preprocessor Options

-C

Keeps comments in the preprocessed output.

-dD

Outputs a list of **#define** or **#undef** for all the macros appear in **#define** and **#undef**, in addition to the normal preprocessed text. When **-E** is not specified, this option is ignored.

-dI

Outputs the **#include** used in the input file, in addition to the normal preprocessed text. When **-E** is not specified, this option is ignored. When **-dM** is specified, this option is ignored.

-dM

Outputs a list of **#define** with macro names and their values for all the macros defined by **#define** or **-D**, instead of the normal preprocessed text. When **-E** is

not specified, this option is ignored.

-dN

Outputs a list of **#define** with macro names for all the macros defined by **#define** or **-D**, in addition to the normal preprocessed text. When **-E** is not specified, this option is ignored.

-D*macro*[=*defn*]

Defines *macro* as the value *defn* as if **#define** does. When =*defn* is omitted, *macro* is defined as decimal constant 1.

-E

Performs preprocessing only and outputs the preprocessed text to the standard output.

-H

Outputs a list of files included by **#include** to the standard error output.

-I*directory*

Adds *directory* to the list of directories searched for files specified by **#include**.

-I-

Specifies that the directories specified with **-**I preceding this option are searched only for the files specified by the form of **#include** "*filename*" and they are not searched for the files specified by the form of **#include** <*filename*>.

-include *file*

Includes *file* at the beginning of the compilation.

-isysroot *directory*

Searches the directory named include under directory for header files specified with **#include**.

-isystem *directory*

Searches *directory* after all the directories specified by **-I** but before the standard system directories.

-M

Outputs a list of the file dependencies instead of the normal preprocessed text.

-MD

Same as **-M -MF** *filename*, where *filename* is a name suffixed by ".d" which is based on input filename or the name specified with **-o** if any.

-MF *filename*

Specifies that the list of the file dependencies is output to *filename* instead of the

default. This option must be specified with **-M**.

-MP

Tells the preprocessor to add a phony target for each dependency output. This option must be specified with **-M**.

-MT *target*

Changes the default target of dependency output to *target*. This option must be specified with **-M**.

-nostdinc

Omits searching the standard system directory for header files.

-P

Omits outputting line directives to preprocessed text.

-U*macro*

Undefines the definition of *macro*.

-undef

Do not predefine any system-specific macros.

-Wp,*option*

Specifies *option* to be passed to preprocessor (cpp). Multiple options or arguments can be specified to this option at once by separating them by commas.

3.11 Assembler Options

-Wa,*option*

Specifies *option* to be passed to assembler (nas). Multiple options or arguments can be specified to this option at once by separating them by commas.

-Xassembler *option*

Specifies an *option* to be passed to assembler (nas). If an option requires an argument, this option must be specified twice, once for the option and once for the argument.

-assembly-list

Outputs assembly list to file. The output filename is a name suffixed by “.O” which is based on input filename.

3.12 Linker Options

-Bdynamic

Enables the linking of dynamic-link libraries at the run-time. This is default when not specifying **-Bstatic**.

-Bstatic

Link user's libraries statically.

-L*directory*

Searches *directory* for libraries specified subsequently to this option, before the directories searched by default.

-l*library*

Specifies a *library* to be linked. Prescribed directories are searched for the library named lib*library*.a.

-nostartfiles

Does not link the standard system startup files.

-nostdlib

Does not link the standard system startup files or libraries.

-rdynamic

Adds all symbols including any unused symbols to the dynamic symbol table at the linking.

-static

Link libraries statically.

-static-nec

Link the NEC SDK libraries statically.

-shared

Generates a shared object.

-Wl,*option*

Specifies *option* to be passed to linker (nld). Multiple options or arguments can be specified to this option at once by separating them by commas.

-Xlinker *option*

Specifies an *option* to be passed to linker (nld). If an option requires an argument, this option must be specified twice, once for the option and once for the argument.

-z *keyword*

Same as nld’s **-z** option.

3.13 Directory Options

--sysroot=*directory*

Specifies a *directory* name where header files and libraries are searched for. The directory named “include” under directory is searched for the header files. The directory named “lib” under directory is searched for the libraries.

-B*directory*

Specifies a *directory* name where commands, header files and libraries are searched for. The specified *directory* is searched for the commands and libraries. The directory named “include” under directory is searched for the header files.

3.14 Miscellaneous Options

--help

Displays usage of the compiler.

-print-file-name=*library*

Displays the full pathname of the library file named *library* which would be linked. When this option is specified, actual compilation and linking are never done.
If the named *library* is not found, only the name specified as *library* is displayed.

-print-prog-name=*program*

Displays the command name named *program* in the compiler system which would be invoked during the compilation through linking. When this option is specified, actual compilation and linking are never done.
If the named command is not found, only the name specified as *program* is displayed.

-noqueue

When the number of licenses exceeds use restriction, the compiler doesn’t stands by until a license is freed.

-v

Displays the invoked commands at each stage of compilation.

--version

Displays the version number and copyrights of the compiler.

3.15 Compiler options which cannot specify by options directive

The following compiler options cannot be specified by options directive.

- Overall Options

  -S, **-c**, **-cf**=*conf*, **-fsyntax-only**, **-o** *file-name*, **-x** *language*, ***@file-name***

- Parallelization Options

  -mno-create-threads-at-startup, **-pthread**

- Code Generation Options

  -no-proginf

- Debugging Options

  -traceback

- Language Options

  -traditional, **-cpp**, **-trigraphs**

- Message Options

  -Werror

- Preprocessor Options

  -C, **-dD**, **-dI**, **-dM**, **-dN**, **-D***macro*[=*defn*], **-E**, **-H**, **-include** *file*, **-M**, **-MD**, **-MF** *filename*, **-MP**, **-MT** *target*, **-P**, **-U***macro*, **-undef**

- Assembler Options

  -Wa,*option*, **-Xassembler** *option*, **-assembly-list**

- Linker Options

  -Bdynamic, **-Bstatic**, **-L***directory*, **-l***library*, **-nostartfiles**, **-nostdlib**, **-rdynamic**, **-static**, **-static-nec**, **-shared**, **-Wl**,*option*, **-Xlinker** *option*, **-z** *keyword*

- Directory Options

  --sysroot=directory, **-B**directory

- Miscellaneous Options

  --help, **-print-file-name**=*library*, **-print-prog-name**=*program*, **-noqueue**, **-v**, **--version**

3.16 Optimization Level and Options' Defaults

The relation between -O*n* and independently optimization options are as follows. Note that -O*n* controls the overall level of optimization, and the same instruction code cannot be created even if an independently optimization option are enabled or disabled are equal. To effectively apply one optimization, optimizations are

interrelated such as applying another ancillary optimizations, and -O*n* controls them to work together. For example specifying the optimization option that is set as the defaults of -O1 with –O0, the instruction code cannot equal to -O1.

Option Name	-O4	-O3	-O2	-O1	-O0
-fargument-alias	-	✓	✓	✓	✓
-fargument-noalias	✓	-	-	-	-
-fassociative-math	✓	✓	✓	-	-
-ffast-math	✓	✓	✓	✓	-
-fignore-induction-variable-overflow	✓	-	-	-	-
-fignore-volatile	✓	-	-	-	-
-finline-copy-arguments	-	✓	✓	✓	✓
-floop-collapse	✓	✓	-	-	-
-floop-fusion	✓	✓	-	-	-
-floop-interchange	✓	✓	-	-	-
-floop-normalize	✓	✓	-	-	-
-floop-strip-mine	✓	✓	-	-	-
-floop-unroll	✓	✓	✓	-	-
-floop-unroll-complete=4	✓	✓	✓	-	-
-fmatrix-multiply	✓	✓	-	-	-
-fmove-loop-invariants	✓	✓	✓	✓	-
-fmove-loop-invariants-if	✓	✓	-	-	-
-fmove-loop-invariants-unsafe	✓	-	-	-	-
-fmove-nested-loop-invariants-outer	✓	✓	✓	✓	-
-fnamed-alias	-	-	-	✓	✓
-fnamed-noalias	✓	✓	✓	-	-
-fouterloop-unroll	✓	✓	-	-	-
-freciprocal-math	✓	✓	✓	-	-
-freplace-loop-equation	✓	-	-	-	-
-fstrict-aliasing	✓	✓	✓	-	-
-fthis-pointer-alias	-	-	-	✓	✓
-fthis-pointer-noalias	✓	✓	✓	-	-

Option Name	-O4	-O3	-O2	-O1	-O0
-msched-none	-	-	-	-	✓
-msched-block	✓	✓	✓	✓	-
-mvector	✓	✓	✓	✓	-
-mvector-dependency-test	✓	✓	✓	-	-
-mvector-fma	✓	✓	✓	-	-
-mvector-merge-conditional	✓	✓	-	-	-

Chapter4 Compiler Directives

This chapter describes the compiler directive of C/C++ compiler. Its format is as follows.

Format:

#pragma _NEC *directive-name* [*clause*]

[no]advance_gather

Allows [Disallows] motion of vector gather instructions in the following loop so that they can be started as advance as possible.

always_inline

A function which includes this directive should be always inlined. This directive must be specified in a called function. A function call which **noinline** is effective is never inlined even if the called function includes this directive. **-O***n*[*n*=2,3,4], **-finline-functions, -fopenmp,** or **-mparallel** is needed to enable this directive.

[no]assoc

Allows [Disallows] associative transformation in which the order of operations may be different from the original.

[no]assume

Allows [Disallows] the use of an array declaration to assume the loop iteration count.

atomic

Specifies that the expression in the statement immediately after atomic is a macro operation which is one of x binop=*expr*, x++, ++x, x-- or --x. See **7.1.5 Forced Loop Parallelization** for details.

cncall

Allows parallelization of a loop which includes function calls.

collapse

Allows parallelization of a loop which includes user defined function calls.

[no]concurrent

Allows [Disallows] automatic parallelization of the following loop. **-mparallel** must be effective. The following schedule-clause whose functionality is the same as OpenMP can be specified.

schedule(static [,*chunk-size*])

schedule(dynamic [,*chunk-size*])

schedule(runtime)

[no]dependency_test

Allows [Disallows] the conditional vectorization by dependency-test.

forced_collapse

Collapses a nested loop forcibly. The user have to guarantee that the loop collapse does not give unexpected result, incorrect result etc.

gather_reorder

Allows the instruction reordering on the assumption that vector loads and vector stores with non-linear subscripts appearing in the following loop do not overlap each other.

[no]inline

A function call in a following statement, a compound statement, an iteration statement, or a selection statement is [not] chosen as a candidate for inlining. **-O***n*[*n*=2,3,4], **-finline-functions, -fopenmp,** or **-mparallel** is needed to enable these directive.

inline_complete

Same as inline. But, if the inlined function includes a function call, the called function is chosen as a candidate for inlining. The inlining applied until there is no function calls if possible. **-O***n*[*n*=2,3,4], **-finline-functions, -fopenmp,** or **-mparallel** is needed to enable this directive.

[no]inner

Allows [Disallows] parallelization of the innermost loop. When it is specified to the innermost loop, it is effective.

[no]interchange

Allows [Disallows] loop interchanging.

ivdep

Regards the unknown dependency as vectorizable dependency during the automatic vectorization. An execution result can be incorrect by vectorizing the loop which is impossible to be vectorized.

[no]list_vector

Allows [Disallows] vectorization of the statement in a loop when an array element with a vector subscript expression appears on both the left and right sides of an assignment operator.

loop_count(*n*)

Assumes loop iteration count as *n* when compiler cannot determine the count by loop controlling expression.

[no]loop_count_test

Allows [Disallows] the conditional vectorization by loop-iteration-count-test.

[no]lstval

loop transformation which does not guarantee the values of the variables in the loop after the loop has been processed.

move / move_unsafe / nomove

move

Allows the loop invariant motion under if-condition.

move_unsafe

Allows the loop invariant motion under if-condition. The unsafe codes which may cause any side effects are moved.

nomove

Disallows the loop invariant motion under if-condition.

nofma

Disallows to use vector fused-multiply-add instruction in the loop.

nofuse

Disallows the loop fusion with the previous loop.

nosync

Parallelizes the loop ignoring unknown dependencies when the array elements in the loop have unknown dependencies.

options "*compiler-option* [*compiler-option*]..."

Specify the compiler options by options directive in the same way as on a command line.

Rules

- The options directive must be specified at the top of your source program.
- Two or more options directives can be specified in succession.
- Blank line, comment line, **#line** and **#ident** can be written before and between options directive.
- The options directive can be specified in the file included by **#include** at the top of your source program.

Remarks:

- An option directive line cannot be continued.
- The directory specified by **-I** in options directive is not searched for reading options directive.
- The upper limits of nesting level of files included by **#include** is 1000.
- The compiler options that control linking or compiler environment cannot be specified. See "3.15 Compiler options which cannot specify by options directive".
- When **-fopenmp**, **-mparallel** and/or **–ftrace** are specified by options directive, they must be specified at linking.

outerloop_unroll(*n*) / noouterloop_unroll

outerloop_unroll(*n*)

Allows outer loop unrolling. The unroll time becomes a power of 2 that is less than or equal to *n*.

noouterloop_unroll

Disallows outer loop unrolling.

[no]packed_vector

Allows to use packed vector instruction in the loop.

parallel for

Applies forced loop parallelization to the loop immediately after this directive. The user must check the validity of the operation when the loop is parallelized. See **7.1.5 Forced Loop Parallelization** for details.

pvreg(*array-name*)

Assign a vector register forcedly to the array "*array-name*" in this routine. The array must satisfy the following conditions.

- Local array
- The type of array must be one of **int**, **unsigned int**, or **float**.
- One-dimensional array
- The number of the array elements is less than or equal to the maximum packed vector length (=512).
- They must be referenced in the packed vectorized loops.
- Their subscript expressions must be the same in all loops.
- The array specified by **vreg** directive cannot be specified by **pvreg** directive.

retain(*array-name*)

Sets higher priority to array "*array-name*" to retain on LLC (Last-Level Cache) in the vectorized loop immediately after this directive.

Note Please specify **-mretain-list-vector** or **-mretain-none** when you use this directive.

select_concurrent

Choose the following loop rather than other loops in a nested loop when applying automatic parallelization.

select_vector

Choose the following loop rather than other loops in a nested loop when applying automatic vectorization.

shortloop

Vectorizes a loop as a short-loop. Compiler assume the iteration count would be less than or equal to the maximum vector register length (=256) when the iteration count is unknown.

[no]shortloop_reduction

Allows [Disallows] the conditional vectorization by iteration count test for a reduction loop.

[no]sparse

sparse

Assumes that the number of mathematical function calling under a conditional expression is only a small number of the total iterations at vectorization.

nosparse

Assumes that the number of mathematical function calling under a conditional expression is a large number of the total iterations at vectorization.

unroll(*n*) / nounroll

unroll(*n*)

Allows loop unrolling. The unroll time is *n*.

nounroll

Disallows loop unrolling.

unroll_complete

Allows loop expansion (complete loop unrolling) of a loop whose iteration count is constant and can be calculated at the compilation.

In order to calculate the number of iterations at compile time, the loop must be of the following form. var*N*(*N*=1,2,...) are assumed to be variables. They shall not be defined in the loop. Furthermore, the types of var*N* and const*N* must be the same. When the value of const3 is 1, the ++ and -- operators may be used in expressions that update the rightmost var1.

- for (var1 = const1; var1 <= const2; var1 = var1 + const3)
- for (var1 = const1; var1 < const2; var1 = var1 + const3)
- for (var1 = const1; var1 >const2; var1 = var1 - const3)
- for (var1 = const1; var1 >= const2; var1 = var1 - const3)
- for (var1 = var2 + const1; var1 <= var2 + const2; var1 = var1 + const3)
- for (var1 = var2 + const1; var1 < var2 + const2; var1 = var1 + const3)
- for (var1 = var2 + const1; var1 > var2 + const2; var1 = var1 - const3)
- for (var1 = var2 + const1; var1 >= var2 + const2; var1 = var1 - const3)
- for (var1 = var2 - const1; var1 <= var2 - const2; var1 = var1 + const3)
- for (var1 = var2 - const1; var1 < var2 - const2; var1 = var1 + const3)
- for (var1 = var2 - const1; var1 > var2 - const2; var1 = var1 - const3)
- for (var1 = var2 - const1; var1 >= var2 - const2; var1 = var1 - const3)

Remark: **unroll_completely** can be used as an alias directive name.

[no]vector

Allows [Disallows] automatic vectorization of the following loop.

vector_threshold(*n*)

Specifies the minimum loop iteration count for vectorization of the following loop.

[no]verror_check

[Not] Checks the value ranges of arguments in the mathematical functions in the vectorized version.

[no]vob

Disallows [Allows] a scalar load, a scalar store or a vector load which is executed after the loop immediately after this directive to overtake the vector store in the loop.

[no]vovertake

Allows [Disallows] all vector stores in the loop are over-taken by the subsequent scalar load, scalar store or vector load.

- An execution result becomes incorrect, if there actually is overlap of areas between an array assignment statement or vector-storing in the loop and scalar-loading, scalar-storing, vector-loading in the loop or behind the loop.
- When it is specified to an outer-loop, it is not effective in the inner loops.

vreg(*array-name*)

Assign a vector register forcedly to the array "*array-name*" in this routine. The array must satisfy the following conditions.

- Local array
- The type of array must be one of **int**, **unsigned int**, **long**, **unsigned long**, **long long**, **unsigned long long**, **float**, or **double**.
- One-dimensional array
- The number of the array elements is less than or equal to the maximum vector length (=256).
- They must be referenced in the vectorized loops.
- Their subscript expressions must have the same subscript values in all loops.
- The array specified by **pvreg** directive cannot be specified by **vreg** directive.

[no]vwork

Allows [Disallows] partial vectorization using loop division. When **novwork** is specified, an outer loop or a loop that contains a nonvectorizable part becomes nonvectorizable as a whole.

Chapter5 Optimization and Vectorization

This chapter describes optimization and automatic vectorization which are useful in making user programs execute quickly.

5.1 Code Optimization

The code optimization eliminates unnecessary operations by analyzing program control and data flow. Where possible, it minimizes the operations involved in a loop and replaces them with equivalent faster operations.

5.1.1 Optimizations

The C/C++ compiler performs the following code optimizations. The parenthesis indicates the options to enable the individual optimizations.

- Common expression elimination (**-O**[*n*] (*n*=1,2,3,4))
- Moving invariant expressions under a conditional expression outside a loop (**-O**[*n*] (*n*=1,2,3,4), **-fmove-loop-invariants**, **-fmove-loop-invariants-unsafe**)
- Simple assignment elimination (**-O** [*n*] (*n*=1,2,3,4))
- Deletion of unnecessary codes (**-O** [*n*] (*n*=1,2,3,4))
- Exponentiation optimization (**-O** [*n*] (*n*=1,2,3,4))
- Converting division to equivalent multiplication (**-O**[*n*](*n*=2,3,4), **-freciprocal-math**)
- Loop fusion (**-O**[*n*] (*n*=3,4))
- Ignoring of **volatile**-qualifier (**-O**[*n*](*n*=4), **-fignore-volatile**)
- Compile-time computation of constant expressions and type conversions (**-O**[*n*] (*n*=1,2,3,4))
- Optimization of complex number computations (**-O**[*n*](*n*=1,2,3,4))
- Removal of unary minus (**-O**[*n*] (*n*=1,2,3,4))
- Optimization of branching (**-O**[*n*] (*n*=1,2,3,4))
- Strength reduction (**-O**[*n*] (*n*=1,2,3,4))
- Removal of an unnecessary instruction to guarantee the last value (**-O**[*n*] (*n*=1,2,3,4))

- In-line expansion of Intrinsic functions (**-O**[*n*] (*n*=1,2,3,4))
- Optimizing by Instruction scheduling (**-msched-***keyword*)

5.1.2 Side Effects of Optimization

- Common expression elimination or code motion may change the points where a calculation is performed. The number of times a calculation is performed also changes the points where errors occur and the number of error occurrences, as compared with the not-optimized object code.
- By moving invariant expressions under a conditional expression outside the loop, expressions which should not be executed are always executed. Therefore an unexpected error and an arithmetic exception may occur.
- When exponentiation optimization is effective, an exception is not detected even if underflow exceptions occur.
- Converting division to equivalent multiplication normally causes a slight error in the result. Although this error can usually be ignored in floating point arithmetic, it may change the result if floating point arithmetic operations are converted to integer arithmetic operations. This conversion can be stopped and avoided by compiler option.
- Optimization by instruction scheduling may produce the following side effect. If a calculation to be executed only when a certain condition is satisfied is moved beyond basic blocks, and it is always executed, an error which should not occur may occur. Also remarkably increases compile time and memory used by the compiler.

5.2 Vectorization Features

5.2.1 Vectorization

Variables and each element of an array are called scalar data. An orderly arranged scalar data sequence such as a line, column, or diagonal of a matrix is called vector data.

Vectorization is the replacement of scalar instructions with vector instructions. In automatic vectorization, the compiler analyzes the source code to detect parts that can be executed by vector instructions.

Automatic vectorization is performed when **-O**[*n*] (*n*=1,2,3,4) is valid.

The compiler option which controls this vectorization is **-mvector**.

The compiler directive option which controls this vectorization is **[no]vector**.

5.2.2 Partial Vectorization

If a vectorizable part and an unvectorizable part exist together in a loop, the compiler divides the loop into vectorizable and unvectorizable parts and vectorizes just the vectorizable part. This vectorization is called partial vectorization.

This vectorization is performed when **-O**[*n*] (*n*=1,2,3,4) is valid.

The compiler option which suppress this vectorization is **-mwork-vector-kind**=none.

The compiler directive option which controls this vectorization is **[no]vwork**.

5.2.3 Macro Operations

Although patterns like the following do not satisfy the vectorization conditions for definitions and references, the compiler recognizes them to be special patterns and performs vectorization by using proprietary vector instructions.

This vectorization is performed when **-O**[*n*] (*n*=1,2,3,4) is valid.

- Sum or inner product

```
S =   S   ±  exp                              (exp: An expression)
```

A sum or inner product that consists of multiple statements is also vectorized.

```
t1=   S   ±  exp1                             (expi: An expression)
t2=   t1  ±  exp2
    ...
S =   tn  ±  expn
```

The compiler option which controls this vectorization is **-mvector-reduction**.

- Product

```
S = S *   exp                                 (exp: An expression)
```

A product that consists of multiple statements is also vectorized.

```
t1 = S    *   exp1                            (expi: An expression)
t2 = t1       *   exp2
    ...
S = tn * expn
```

The compiler option which controls this vectorization is **-mvector-reduction**.

- Iteration

```
a[i]  =  exp1   ±  a[i-1];              (expi: An expression)
a[i]  =  exp1   *  a[i-1];
a[i]  =  exp1   ±  a[i-1]  *  exp2;
a[i]  =  (exp1  ±  a[i-1]) *  exp2;
```

An iteration consists of multiple statements and is also vectorized.

```
t =   exp1 ± a[i-1];                    (expi: An expression)
a[i] = t * exp2;
```

The compiler option which controls this vectorization is **-mvector-iteration** and **-mvector-iteration-unsafe**.

- Maximum values and minimum values
  - Finding the maximum or minimum value only

    Example:

```
for (i = 0; i < n; i++) {
    if (a[i] > amx)
        amx = a[i];
}
```

  - Finding the maximum or minimum value and the value of its subscript expression

    Example:

```
for (i = 0; i < n; i++) {
    if (a[i] > amx) {
        amx = a[i];
        ix = i;
    }
}
```

  - Finding the maximum or minimum value, the values of its subscript expressions, and other values

Example:

```
for (j = 0; j < n; j++) {
    for (i = 0; i < n; i++) {
        if (a[i][j] > amx) {
            amx = a[i][j];
            ix = i;
            iy = j;
        }
    }
}
```

- Search

  A loop that searches for an element that satisfies a given condition is vectorized.

  Example:

```
for (i = 0; i < n; i++) {
    if (a[i] == 0)
        break;
}
```

  All of the following conditions must be satisfied.

  - This is the innermost loop.
  - There is just one branch out of the loop.
  - The condition for branching out of the loop depends on repetition of the loop.
  - There must not be an assignment statement to an array element or an object pointed to by a pointer expression before the branch out of the loop.
  - All basic conditions for vectorization are satisfied except for not branching out of the loop.

- Compression

  A loop for compressing elements that satisfy a given condition is vectorized.

  Example:

```
j = 0;
for (i = 0; i < N; i++) {
    if (x[i] >= 0.0) {
        j = j + 1;
        y[j] = z[j];
    }
}
```

- Expansion

  A loop for expanding values to elements that satisfy a given condition is vectorized.

  Example:

```
j = 0;
for (i = 0; i < N; i++) {
    if (x[i] >= 0.0) {
        j = j + 1;
        z[j] = y[j];
    }
}
```

5.2.4 Conditional Vectorization

The compiler generates a variety of codes for a loop, including vectorized codes and scalar codes, as well as special codes and normal codes. The type of code is selected by run-time testing at execution when conditional vectorization is performed. Run-time testing are following.

- Data dependency
- Loop iteration count
- Loop iteration for reduction operation

This vectorization is performed when **-O**[*n*] (*n*=2,3,4) and **–mvector** is valid.
The compiler option which controls this vectorization is following.

- Conditional vectorization with data dependency is **-mvector-dependency-test**.
- Conditional vectorization with loop iteration count is **-mvector-loop-count-test**.
- Conditional vectorization with loop iteration for reduction operation is **-mvector-shortloop-reduction**.

The compiler directive option which controls this vectorization is following.

- Conditional vectorization with data dependency is **dependency_test**.
- Conditional vectorization with loop iteration count is **loop_count_test**.
- Conditional vectorization with loop iteration for reduction operation is **[no]shortloop_reduction**.

5.2.5 Outer Loop Strip-mining

When the iteration count of a loop is greater than the maximum-vector-register-

length (=256), the compiler puts a loop around the vector loop, which splits the total vector operation into "strips" so that the vector length will not be exceeded.
When there are references of array elements whose subscript expressions do not include the induction variables of the outer loop in the inner loop of a tightly nested loop, the inner loop is split into a strip loop and the strip loop is moved outside of the outer loop so that invariants can be kept in the vector register.
This optimization is performed when **-O**[*n*] (*n*=3,4) is valid.
The compiler option which controls this vectorization is **-floop-strip-mine**.

Note In tightly nested loops, the loops nested together must look as shown in Example 1. In this case, there is no executable statement between the inner and outer loops.

Example: Tightly nested loop

```
for (k = 0; k < 10; k++) {
    for (j = 0; j < 20; j++) {
        for (i = 0; i < 30; i++) {
            a[k][j][i] = b[k][j][i] * c[k][j][i];
        }
    }
}
```

Example: Not tightly nested loop

```
for (k = 0; k < 10; k++) {
    for (j = 0; j < 20; j++) {
        for (i = 0; i < 30; i++) {
            a[k][j][i] = b[k][j][i] * c[k][j][i];
        }
        x[j][k] = y[j][k] + z[j][k];
    }
}
for (k = 0; k < 10; k++) {
    for (j = 0; j < 20; j++) {
        for (i = 0; i < 30; i++) {
            a[k][j][i] = b[k][j][i] * c[k][j][i];
        }
        for (i = 0; i < 30; i++) {
            s[k][j][i] = t[k][j][i] * u[k][j][i];
        }
    }
}
```

5.2.6 Short-loop

A loop code which does not have "terminate loop?" is generated for a loop whose iteration count is less than or equal to the maximum-vector-register-length (=256). This kind of loop is called a "short-loop".

This optimization is performed when **-O**[*n*] (*n*=1,2,3,4) is valid.

The compiler directive option which controls this optimization is **shortloop**.

5.2.7 Packed vector instructions

A packed data is packed two 32bit data in each element of a vector register. Packed vector instructions calculates a packed data. Packed vector instructions can calculate twice the data of vector instructions by one instruction.

The compiler option which controls using packed vector instructions is **-mvector-packed**.

The compiler directive option which controls using packed vector instructions is **[no]packed_vector**.

5.2.8 Other

Deletion of common expression, deletion of simple assignments, deletion of unnecessary codes, conversion of division to equivalent multiplication and removal of an unnecessary instruction to guarantee the last value are also performed for vectorized codes.

Additionally the following optimizations are performed for vectorized codes. The parenthesis indicates the options to enable the individual optimizations.

- Extracting scalar operations (**-O**[*n*] (*n*=1,2,3,4))
- Vectorization by Statement Replacement (**-O**[*n*] (*n*=1,2,3,4))
- Loop collapse (**-O**[*n*] (*n*=3,4), **-floop-collapse**)
- Outer loop unrolling (**-O**[*n*] (*n*=3,4), **-fouterloop-unroll**)
- Loop rerolling (**-O**[*n*] (*n*=3,4))
- Recognization matrix multiply loop (**-O**[*n*] (*n*=3,4), **-fassociative-math**, **-fmatrix-multiply**)
- Loop expansion (**-O**[*n*] (*n*=2,3,4), **-floop-unroll-complete**=*m*)

5.2.9 Remarks on Using Vectorization

- The execution result of the summation, the inner product, the product and the iteration may differ before and after vectorization because the order of their operations may differ before and after vectorization.
- The 8 byte integer iteration is vectorized by using a floating-point instruction. So when the result exceeds 52 bits or when a floating overflow occurs, the result differs from that of scalar execution.
- To increase speed, the vector versions of mathematical functions do not always use the same algorithms as the scalar versions.
- Optimization techniques, such as conversion of division to multiplication, are applied differently.
- Optimization techniques, such as reordering of arithmetic operations, are applied differently.
- The detection of errors and arithmetic exceptions by intrinsic functions may differ before and after vectorization.
- When the compiler checks whether vectorization would preserve the proper dependency between array definitions and references, it assumes that all values of subscript expressions are within the upper and lower limits of the corresponding size in the array declaration. If a loop violating this condition is vectorized, correct results are not guaranteed.
- When a loop containing **if** statement, **switch** statement, or a conditional operator is vectorized, arithmetic operations are carried out only for the part that conditionally requires them, but arrays are referenced as many times as the iteration count called for by the loop structure and array elements that should not be referenced are referenced. Unless the arrays have enough area reserved to satisfy the iteration count, memory access exceptions can occur as a result.
- When a loop containing a branch out of the loop is vectorized, arithmetic operations are carried out unconditionally for the part before the branch point, as many times as the iteration count called for by the loop structure. Therefore, arithmetic operations that should not be carried out are carried out, or data that should not be referenced are referenced. These events can cause errors or exceptions.

- The alignment size of vectorizable data must be same as size of the data type (4 bytes or 8 bytes). When a loop containing array elements or objects pointed to by the pointer expressions which do not satisfy the vectorizable alignment conditions is vectorized, errors or exceptions may occur. In such a case, specify **#pragma _NEC novector** before the loop or **-mno-vector** to stop vectorization. The following data may not satisfy the vectorizable alignment conditions.
  - Arguments
  - Objects pointed to by the pointer

  The compiler assumes these data to satisfy the alignment condition, and vectorizes the loop.

Chapter6 Inlining

6.1 Automatic Inlining

When automatic inlining is enabled, the compiler chooses the appropriate functions by analyzing the source files, and inlines them automatically.
The compiler option which controls this optimization is **-finline-functions**.

6.2 Explicit Inlining

6.2.1 Description

When using the explicit inlining, an inlining directive which controls inlining must be specified before a statement, a compound statement, an iteration statement, or a selection statement including inlined function calling. The compiler option **-finline-functions** is not needed, but -O*n*[*n*=2,3,4], -finline-functions, -fopenmp, or **-mparallel** is needed.
The compiler has the following directives for explicit inlining.

- **always_inline**
  A function which includes this directive should be always inlined. This directive must be specified in a called function. A function call has **noinline** is never inlined even if the called function includes this directive.
- **inline**
  A function call in a following statement, a compound statement, an iteration statement, or a selection statement is chosen as a candidate for inlining.
- **inline_complete**
  Same as inline. But, if the inlined function includes a function call, the called function is chosen as a candidate for inlining. The inlining applied until there is no function calls if possible.
- **noinline**
  A function call in a following statement, a compound statement, an iteration statement, or a selection statement is never inlined. The function which includes **always_inline** is not inlined, too.

6.2.2 Specifying Inline Directive

(1) Called function

always_inline must be specified in a called function.

```
double small_func(double a)
{
#pragma _NEC always_inline
    return sqrt(a);
}
```

(2) Statement

inline / **inline_complete** / **noinline** affect all function calls in a following statement.

```
#pragma _NEC inline
    x = func1(a) + func2(a)
    x += func3(a);
```

func1() and func2() are candidates for inlining, but func3() is not.

(3) Block

inline / **inline_complete** / **noinline** affect all function calls in a following block.

```
#pragma _NEC inline
    {
        func1();
        func2();
    }
```

func1() and func2() are candidates for inlining.

(4) Loop

inline / **inline_complete** / **noinline** affect all function calls in a following **for** loop, **while** loop, or **do-while** loop.

```
#pragma _NEC inline
    for (i = ifunc(); i < 1000; i++) {
        z[i] = func1();
        w[i] = func2();
    }
```

ifunc(), func1(), and func2() are candidates for inlining.

(5) **if**-statement and **switch**-statement

inline / **inline_complete** / **noinline** affect all function calls in a following **if** statement, **switch** statement, and their sub-statements.

```
#pragma _NEC inline
    if (ifunc1()) {
        x = func1();
    }
    else if (ifunc2()) {
        x = func2();
    }
    else {
        x = func3();
    }
```

ifunc1(), ifunc2(), func1(), func2(), and func3() are candidates for inlining.

6.2.3 Remarks

- **always_inline**, **inline**, **inline_complete**, and **noinline** are effective when **-O***n* [*n*=2,3,4], **-finline-functions**, **-fopenmp**, or **-mparallel** are enabled.
- **always_inline** is ignored when both __attribute__((noinline)) and **always_inline** are specified.
- The function definition which includes **always_inline** is not removed. Be careful that the function definition is removed when __attribute__((always_inline)) is specified.
- A function call which **noinline** is effective is not inlined even if the called function includes **always_inline**.
- A block includes a block and each block has opposite directive, the immediately before directive is effective for the inner block.

```
#pragma _NEC inline
    {
        x = func1();            // Candidate for inlining
#pragma _NEC noinline
        {
            y = func2();        // Not inlined
        }
    }
```

- A function call in an initializer in a declaration is not inlined even if **inline** is effective.

6.3 Cross-file Inlining

The compiler inlines functions included in source files other than a source file of the compilation target. This inlining is called cross-file inlining.
Cross-file inlining is enabled when automatic inlining is enabled and source files to search for functions to inline are specified.
The following examples show how to specify the source files.

- A source file is specified.

```
$ ncc -c -finline-functions -finline-file=sub.c call.c
```

- A source file and all input source files are specified.

```
$ ncc -c -finline-functions -finline-file=sub2.c:all call.c sub.c
```

- All source files under a directory are specified.

```
$ ls dir
sub.c sub2.c sub3.c
$ ncc -c -finline-functions -finline-directory=dir sub.c
```

- All source files under a directory except for a specific source file are specified.

```
$ ls dir
sub.c sub2.c sub3.c
$ ncc -c -finline-functions -finline-directory=dir -fno-inline-file=sub2.c
call.c
```

IL files can be also specified as files to search. Compilation time can become shorter when you specify IL files instead of source files.

- An IL file is generated and specified.

```
$ ncc -mgenerate-il-file sub.c
$ ncc -c -finline-functions -mread-il-file sub.cil main.c
```

6.4 Inline Expansion Inhibitors

Expansion inhibitors are used when one of the following conditions occurs.

- The function to be inlined cannot be located.
- The arguments used in the calling sequence do not match the arguments in the function to be inlined.
- There is a conflict between **unions** of the calling function and the function to be inlined.
- A function name referenced in the function to be inlined conflicts with a nonfunction name used in the calling function.
- The function to be inlined contains OpenMP directives.
- The function to be inlined contains a recursive call of it.

6.5 Notes on Inlining

- If inlining is applied to too many functions in a program, the volume of the codes may increase, causing the instruction cache to overflow and the performance of the program to decrease. Choose the functions to be inlined carefully.
- A function called recursively cannot be inlined.
- In cross-file inlining, if large or many programs are searched, the compilation time can become long or memory used at the compilation may increase.
- In cross-file inlining, whether routines are inlined or not may change by the compilation order, because the compiler does not search the source files and continues the compilation when modules referred in programs of source files specified by **-finline-file** or **-finline-directory** are not found. Specify **-finline-abort-at-error** when you want to stop the compilation at the case.
- Cross-file inlining can be used only in C language.

Chapter7 Parallelization

7.1 Automatic Parallelization

7.1.1 Description

The compiler automatically detects the parallelism of loop iterations and statement groups, transforms a program to enable it to be executed in parallel, and generates parallelization control structures when automatic parallelization is enabled.
The compiler option which controls this optimization is **-mparallel**.

7.1.2 Conditional Parallelization Using Threshold Test

Parallelization can slow down execution if the loop contains insufficient work to compensate for the added overhead.
If the loop nest iteration count cannot be determined at compilation, the automatic parallelization function generates codes to execute a threshold test at run time. If it is calculated at run time that the loop has a lot of work, the loop is executed in parallel mode. Otherwise the loop is executed serially. This parallelization is called parallelization using a workload threshold test.
Automatic parallelization adjusts the threshold value based on the iteration count of the loop and the number/type of operations in each loop. At run time, the iteration count of the loop and the threshold value are compared. If the iteration count is larger than the threshold value, the parallelized loop is executed. Otherwise, the non-parallelized loop is executed.
The compiler option which controls this optimization is **-mparallel-threshold**=*n*.

7.1.3 Conditional Parallelization Using Dependency Test

If a loop is suitable for parallelization except that it is potentially dependent, automatic parallelization may generate an if-then block in the same way as for parallelization using a threshold test. When evaluated at run time, this test determines whether the loop can execute correctly on multiple tasks, or must be run on a single task. For single loops and double-nested loops, this test is combined with a threshold test.

7.1.4 Parallelization of inner Loops

When no outer loop can be parallelized, inner loops are analyzed for parallelization

operations. However, inner loops that clearly exceed the threshold value are automatically parallelized even if inner loops are not requested.
The compiler option which controls this optimization is **-mparallel-innerloop**.

7.1.5 Forced Loop Parallelization

#pragma _NEC parallel for parallelizes a **for**-loop that is not parallelized by the compiler but the user knows that it can be parallelized. The user must check the validity of the operation when the loop is parallelized.

The for-statements must be in the form of "**for** (*init-expr*; *var relational-op b*; *incr-expr*)". The terms of the directive must fulfill the following conditions:

- *init-expr* must be one of "*var*=*lb*" or "integer-type *var*=*lb*".
- incr-expr must be one of ++var, var++, --var, var--, var+=incr, var-=incr, var=var+incr or var=var-incr.
- *var* is a scalar variable whose type is **int**, **long**, **long int**, **long long** or **long long int**.
- *relational-op* is one of **<**, **<=**, **>=**, **>** or **!=**.
- *lb*, *b* and *incr* must be loop invariant expressions.

The following **schedule**-clause whose functionality is the same as OpenMP can be specified.

schedule(static [,*chunk-size*])
schedule(dynamic [,*chunk-size*])
schedule(runtime)

Additionally, **private**-clause whose functionality is the same as OpenMP can be specified.

private(scalar-variable[,scalar-variable]...)

#pragma _NEC atomic must be specified when a statement immediately after **atomic** is a macro operation which is one of *x binop*=*expr*, *x*++, ++*x*, *x*-- or --*x*.
The statement must fulfill the following conditions:

- *x* must be a scalar variable which can be stored as a value.
- *expr* must be a scalar expression in which *x* does not appear.
- *binop* must be one of +, *, -, /, &, ^, |, << or >>. It must not be overloaded.

The following code is an example inserting forced-loop parallelization directives.

Example:

```
double sub (double *a, int n)
{
    int i, j;
    double b[n];
    double sum = 1.0;
    ...
#pragma _NEC parallel for schedule(dynamic,16)
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
#pragma _NEC atomic
            sum += a[j] + b[i];
        }
    }
    ...
    return sum;
}
```

7.2 OpenMP Parallelization

7.2.1 Using OpenMP Parallelization

Specify **-fopenmp** to use OpenMP parallelization at compilation and linking. See the OpenMP specifications for OpenMP directives and remarks.

Example: Inserting an OpenMP directive

```
double sub (double *a, int n)
{
    int i, j;
    double b[n];
    double sum = 1.0;
    ...
#pragma omp parallel for reduction(+:sum)
    for (j = 0; j < n; j++) {
        for (i = 0; i < n; i++) {
            sum += a[j] + b[i];
        }
    }
    ...
    return sum;
}
```

7.2.2 OpenMP 5.0 Parallelization

The following features of OpenMP Version 5.0 are supported.

- **loop** construct
- **parallel loop** construct
- **parallel master construct**
- OMPT interface

7.2.3 Extensions on OpenMP Parallelization

The environment variables of OpenMP Version 4.5 whose name are prefixed with "VE_" are also supported. If both environment variables with and without "VE_" are specified, the value which is specified by the environment variable prefixed by "VE_" is applied.

Example: Specify the environment variables (applied **VE_OMP_NUM_THREADS**)

```
$ export OMP_NUM_THREADS=4
$ export VE_OMP_NUM_THREADS=8
```

7.2.4 Restrictions on OpenMP Parallelization

The features of OpenMP Version 5.0 except for listed in section 7.2.2 is restricted.
The following features of OpenMP Version 4.5 is restricted.

- All directives/clauses described in "Device Constructs"
  Compiler does not generate any device code and target regions run on the host
- All syntax described in "Array Sections" except in **reduction** clause
- All directives/clauses described in "Cancellation Constructs"
- All directives/clauses described in "Controlling OpenMP Thread Affinity"
- **distribute**, **target**, **teams**
  distribute, **target** and **teams** in directives for combined construct and all clauses related to them are ignored.
  Example : "**target parallel for**" is treated as "**parallel for**".
- **taskloop** constructs
- **parallel for simd** construct and **for simd** construct
  Treated as **parallel for** and **for** respectively
- **simd** construct
  If **saflen** clause or **simdlen** clause is not specified, treated as **ivdep** directive.
- **declare reduction** construct

- **allocate** clause
- **bind** clause
- **if** clause with directive-name-modifier
- **in_reduction**,**task_reduction** clause
- **ordered** clause with *parameter*
- **schedule** with *modifier*
- **depend** clause with array variable
- **depend** clause with **source** or **sink** of *dependence-type*
- **critical** construct with **hint**
- **atomic** construct with **seq_cst**
- **linear** clause with *modifier*
- nested parallelism

7.3 Threads

7.3.1 Set and Get Number of Threads

In automatic parallelized programs, parallel processing is realized based on OpenMP parallel functions. Therefore, you can set the number of threads at execution by the environment variable **OMP_NUM_THREADS** or **VE_OMP_NUM_THREADS** in automatic parallelized and OpenMP parallelized programs.
OpenMP runtime library routines can set and get the number of threads at execution in automatic parallelized programs.

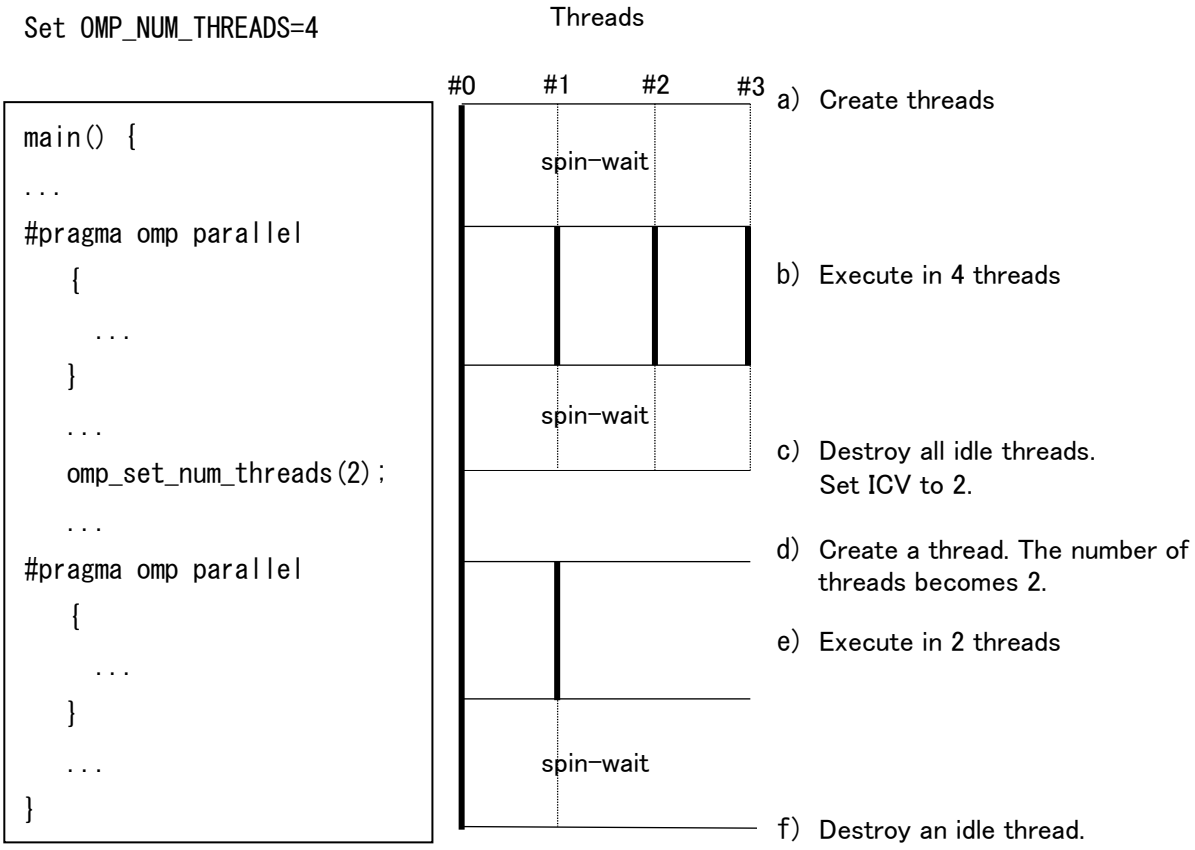
```
extern void omp_set_num_threads(int num_threads);   // Set number of threads
extern int omp_get_num_threads(void);  // Get number of threads
extern int omp_get_max_threads(void);  // Get upper bounds on number of threads
extern int omp_get_thread_num(void);   // Get thread number
```

The number of threads at execution is the same as the number of available VE cores if it is not set by the environment variable **OMP_NUM_THREADS** or **VE_OMP_NUM_THREADS** before the program execution.

7.3.2 Thread Creation and Destroy

In automatic parallelized and OpenMP parallelized programs, the threads are created

before the routine **main**(), and they are destroyed at the program termination. The following figure shows how threads are created and destroyed. Assume that the environment variable **OMP_NUM_THREADS** is set to 4.



(a) Three idle threads are created by master thread (#0) before main() starts. The idle threads are spin-waiting and wait for the task to be assigned by the master thread.

(b) Tasks are assigned to the threads by master task at the entry of parallel region, and it is executed in four threads. At the end of parallel region, three threads are spin-waiting and wait for the task to be assigned by the master thread again.

(c) At the calling of omp_set_num_threads(2), all idle threads are destroyed and set ICV to 2.

(d) A thread is created at the entry of the next parallel region.

(e) The parallel region is executed in two threads.

(f) The idle thread is destroyed at the end of program execution.

Note When outputting execution analysis information an auto-

parallelized program using PROGINF and FTRACE, keep the following in check:

- The number of operations for the spin-waiting of the thread created before main program starts is added in PROGINF, but not in FTRACE.
- In PROGINF, the “Vector Operation Ratio” may decrease. This is due to calculating the displayed value in PROGINF from the counter of the whole process which includes the number of operations for the spin-waiting of the thread created before main program starts.

See the manual ”PROGINF/FTRACE User’s Guide” for the detail of PROGINF or FTRACE.

7.3.3 Postpone Thread Creation

By default, idle threads are created before the routine **main**(). It can be change at the first parallel region by the following compiler option at linking.

```
$ ncc -fopenmp -mno-create-threads-at-startup -static-nec a.o
$ ncc -mparallel -mno-create-threads-at-startup -static-nec b.o
```

7.4 Notes on Using Parallelization

- After parallelization, the total CPU time is increased due to the overhead of parallelization.
- When parallelizing a function that includes function calls, the inside of the called function must be checked to see if the definition and/or reference of shared data is valid.
- Automatic parallelization is applied to the loops outside of a parallel region of OpenMP when **-fopenmp** and **-mparallel** are specified at once. If you don't want to apply automatic parallelization to a routine containing OpenMP directives, specify **-mno-parallel-omp-routine**.

Chapter8 Compiler Listing

This chapter describes the output lists of the C/C++ compiler.

The compilation list is created in the current directory, under the name "*source-file-name*.L".

8.1 Option List

An option list is output when **-report-option** or **-report-all** is specified.

Format:

```
NEC C/C++ Compiler (3.0.7) for Vector Engine     Thu Jun 18 10:18:05 2020  (a)

FILE NAME: fft.c     (b)

  COMPILER OPTIONS : -report-option    (c)

OPTIONS DIRECTIVE: -O4   (d)

PARAMETER :

Optimization Options :
  (e)                                          (f)
-On                                          : 4
-fargument-alias                             : disable
-fargument-noalias                           : enable
-fassociative-math                           : enable
```

(a) Compiler revision and compilation date

(b) Name of source file

(c) Compiler options which specify by command line

(d) Compiler options which specify by options directive

(e) Compiler option

(f) Value of Compiler option

8.2 Diagnostic List

A diagnostic list is output when **-report-diagnostics** or **-report-all** is specified.

8.2.1 Format of Diagnostic List

The format of the diagnostic list is as follows.

Format:

```
NEC C/C++ Compiler (1.0.0) for Vector Engine     Wed Jan 17 14:55:20 2018  (a)

FILE NAME: fft.f90              (b)

FUNCTION NAME: FFT_3D    (c)
DIAGNOSTIC LIST

 LINE              DIAGNOSTIC MESSAGE
    (d)       (e)       (f)
     7: inl(1222): Inlined
     9: vec( 101): Vectorized loop.
```

(a) Compiler revision and compilation date

(b) Name of source file

(c) Name of function that includes loops or statements corresponding to diagnostic

(d) Line number

(e) Kind of Diagnostic and message number

Kind of Diagnostic is as follows.

vec : Vectorization diagnostic

opt : Optimization diagnostic

inl : Inlining diagnostic

par : Parallelization diagnostic

(f) Diagnostic message

8.2.2 Notes

- A diagnostic message for a statement and a loop in an inlined function is not output in a diagnostic list for a function that calls the inlined function. Refer to the diagnostic list for the inlined function when you need to refer to its diagnostic messages.

8.3 Format List

A format list is output when **-report-format** or **-report-all** is specified. The source lines for each function together with the following information are output to the list.

- The vectorized status of loops.
- The parallelized status of loops.

- The status of inline expansion

8.3.1 Format of Format List

The format of the format list is as follows.

```
NEC C/C++ Compiler (1.0.0) for Vector Engine     Wed Jan 17 14:55:16 2018   (a)

FILE NAME: a.c       (b)

FUNCTION NAME: func  (c)
FORMAT LIST

 LINE   LOOP      STATEMENT
    (d)   (e)        (f)
     1:           int func(int m, int n)
     2:           {
     3:               int i,j, a[m][n], b[m][n];
     4: +------>      for (i = 0; i < m; i++) {
     5: |V----->          for (j = 0; j < n; j++) {
     6: ||                    a[i][j] = a[i][j] + b[i][j];
     7: |V-----           }
     8: +------       }
     9:               return a[0][0];
    10:           }
```

(a) Compiler revision and compilation date

(b) Name of source file

(c) Name of function

(d) Line number.

(e) Vectorization and parallelization status of each loop and inlining status of function calls

(f) Corresponding source file line

8.3.2 Loop Structure and Vectorization/Parallelization/Inlining Statuses

The following examples show how the loop structure and vectorization, parallelization and inlining statuses are output.

- The whole loop is vectorized.

```
V------> for (i = 0; i < n; i++) {
|
V------  }
```

- The loop is partially vectorized.

```
S------> for (i = 0; i < n; i++) {
|
S------  }
```

- The loop is conditionally vectorized.

```
C------> for (i = 0; i < n; i++) {
|
C------  }
```

- The loop is parallelized.

```
P------> for (i = 0; i < n; i++) {
|
P------  }
```

- The loop is parallelized and vectorized.

```
Y------> for (i = 0; i < n; i++) {
|
Y------  }
```

- The loop is not vectorized.

```
+------> for (i = 0; i < n; i++) {
|
+------  }
```

- The nested loops are collapsed and vectorized.

```
W------> for (i = 0; i < n; i++) {
|*------>     for (j = 0; j < m; j++)
||
|*-----       }
W------  }
```

- The nested loops are interchanged and vectorized.

```
X------> for (i = 0; i < n; i++) {
|*------>     for (j = 0; j < m; j++) {
||
|*-----       }
X------  }
```

- The outer loop is unrolled and inner loop is vectorized.

```
U------> for (i = 0; i < n; i++) {
|V------>     for (j = 0; j < m; j++) {
||
|V-----       }
U------  }
```

- The loops are fused and vectorized.

```
V------> for (i = 0; i < n; i++) {
|
|        }
|        for (i = 0; i < n; i++) {
|
V------  }
```

- The loop is expanded.

```
*------> for (i = 0; i < 4; i++) {
|
*------  }
```

- A character in the 17th column indicates how the line is optimized.
  - "I" indicates that the line includes a function call which is inlined.
  - "M" indicates that the nested loop which includes this line is replaced with vector-matrix-multiply routine.
  - "F" indicates that a fused-multiply-add instruction is generated for an expression in this line.
  - "R" indicates that **retain** directive is applied to an array in this line.
  - "G" indicates that a vector gather instruction is generated for an expression in this line.
  - "C" indicates that a vector scatter instruction is generated for an expression in this line.
  - "V" indicates that **vreg** directive or **pvreg** directive is applied to an array in this line.

8.3.3 Notes

- The loop structure or vectorization / parallelization statuses may be incorrect when a part of the loop is included in a header file.
- The loop structure or vectorization / parallelization statuses may be incorrect when two or more loops are written in a line.
- The format list of a function is not output when the entry point of the function is included in a header file.
- When a line after a loop is a pre-processor directive line, it is treated as the end of the loop as follows.

```
V------> for (i = 0; i < n; i++) {
|
|        }
V------  #if 0
```

8.4 Optimization List of Each Module

An optimization list of inlining module, vectorization module and code generation module is output.

8.4.1 Inlining Module

An optimization list of inlining module is output when **-report-inline** or **-report-all** is specified.

Format:

```
NEC C/C++ Compiler (3.1.0) for Vector Engine     Thu Sep 17 07:33:16 2020  (a)

FILE NAME: fft.c        (b)

FUNCTION NAME: func3    (c)

INLINE LIST

    INLINE REPORT: func3 (fft.c:17)
    (d)
    -> INLINE: func2 (fft.c:19)              (e)
       -> NOINLINE: func0 (fft.c:12)         (e)
          *** Source for routine not found.  (f)
       -> INLINE: func1 (fft.c:13)           (e)
```

(a) Compiler revision and compilation date

(b) Name of source file

(c) Name of procedure

(d) Level of procedures to be inlined from the bottom of the calling tree.

(e) Inlining status of procedure calls

(f) Diagnostic message

8.4.2 Vectorization Module

An optimization list of vectorization module is output when **-report-vector** or **-report-all** is specified.

Format:

```
NEC C/C++ Compiler (3.1.0) for Vector Engine    Thu Sep 17 08:10:39 2020  (a)

FILE NAME: vec.c        (b)

FUNCTION NAME: func     (c)

VECTORIZATION LIST

  LOOP BEGIN: (vec.c:3)
    <Unvectorized loop.>            (d)

    LOOP BEGIN: (vec.c:4)
      <Vectorized loop.>            (d)
      *** The number of VGT,   VSC.     :  0,  0. (vec.c:4)          (e)
      *** The number of VLOAD, VSTORE.  :  1,  1. (vec.c:4)          (e)
    LOOP END
  LOOP END
```

(a) Compiler revision and compilation date

(b) Name of source file

(c) Name of procedure

(d) Vectorization status of each loop

(e) Diagnostic message

8.4.3 Code Generation Module

An optimization list of code generation module is output when **-report-cg** or **-report-all** is specified.

Format:

```
NEC C/C++ Compiler (3.1.0) for Vector Engine    Thu Sep 17 08:10:39 2020  (a)

FILE NAME: vec.c        (b)

FUNCTION NAME: func     (c)

CODE GENERATION LIST

  Hardware registers    (d)
    Reserved                  : 10 [sl fp lr sp s12 s13 tp got plt s17]
    Callee-saved              : 16 [s18-s33]
    Assigned
      Scalar registers        : 32 [s0-s12 s15-s16 s18-s21 s23-s32 s61-s63]
      Vector registers        : 35 [v0 v30-v63]
      Vector mask registers   :  0
      VREG directive          :  2 [v18-v19]

  Routine stack         (e)
    Total size                : 256 bytes
      Register spill area     :  16 bytes
      Parameter area          :  40 bytes
      Register save area      : 176 bytes
      User data area          :  16 bytes
      Others                  :   8 bytes

    Note: Total size of Routine stack does not include
          the size extended by alloca() and so on.


  LOOP BEGIN: (vec.c:3)
    LOOP BEGIN: (vec.c:4)
      *** The number of VECTOR REGISTER SPILL  (f)
            Total                                   : 14
              Across calls                          : 11
              Not enough registers                  : 1
              Over basic blocks                     : 1
              Others                                : 1
      *** The number of VECTOR REGISTER RESTORE
            Total                                   : 14
              Across calls                          : 11
              Not enough registers                  : 1
              Over basic blocks                     : 1
              Others                                : 1
      *** The number of VECTOR REGISTER TRANSFER    : 12

      *** The number of SCALAR REGISTER RESTORE
```

```
          Total                                  : 14
            Across calls                         : 11
            Not enough registers                 : 1
            Over basic blocks                    : 1
            Others                               : 1
    *** The number of SCALAR REGISTER RESTORE
          Total                                  : 14
            Across calls                         : 11
            Not enough registers                 : 1
            Over basic blocks                    : 1
            Others                               : 1
    *** The number of SCALAR REGISTER TRANSFER   : 21
  LOOP END
LOOP END
```

(a) Compiler revision and compilation date

(b) Name of source file

(c) Name of procedure

(d) Number of registers used for each type of register information

Reserved	: System reserved registers
Callee-saved	: Registers that save across procedure calls
Assigned	: Registers assigned to calculations and user data

(e) Stack information

Register spill area	: Stack area for register spill
Parameter area	: Stack area for parameter area
Register save area	: Stack area for register save area
User data area	: Stack area for user data area
Others	: Others

(f) Cause of register spill, restore and transfer for each loop

Across calls	: Because it across procedure calls
Not enough registers	: Because the registers are shortage
Over basic blocks	: Because it is used across the basic blocks
Others	: Others

Chapter9 Programming Notes Depending on the Language Specification

9.1 Builtin Functions

9.1.1 Performance Tuning Support

```
void __builtin_vprefetch(const void *target, size_t size)
```

Prefetches specified *size* of data started from the address *target*.

9.1.2 Debugging Support

```
void __builtin_traceback(unsignd long *framepointer)
```

Outputs traceback information when the environment variable **VE_TRACEBACK** is set.

Example:

```
__builtin_traceback((unsigned long *)__builtin_frame_address(0));
abort();
```

9.2 Implementation-Defined Specifications

9.2.1 Data Types

9.2.1.1 Size and alignment

The following table shows the data types available in the C/C++ compiler and their size and alignment. [Unit: Bytes]

Type	Size	Alignment	Description
_Bool	1	1	Available in C.
bool	1	1	Available in C++.
char **signed char** **unsigned char**	1	1	**char** means **signed char** by default. It can be changed to **unsigned char** by **-funsigned-char**.
short **short int** **unsigned short** **unsigned short int**	2	2	

Type	Size	Alignment	Description
int **unsigned int**	4	4	
long **long int** **unsigned long** **unsigned long int**	8	8	
long long **long long int** **unsigned long long** **unsigned long long int**	8	8	
float	4	4	Single-precision real type.
double	8	8	Double-precision real type.
long double	16	16	Quadruple-precision real type.
float _Complex	8	4	Single precision complex type.
double _Complex	16	8	Double precision complex type.
long double _Complex	32	16	Quadruple precision complex type.
pointer	8	8	
enum	4	4	Corresponds with int type.
	8	8	Corresponds with long type.

9.2.1.2 Size and alignment of derived type

The derived type is constructed from fundamental types. They are classified into the array type, structure type, union type, pointer type, and function type in C language.

(1) Array type

- Size

  (size-of-the-element) * (number-of-the-elements)

- Alignment

  Requires the same size and alignment as the array element.

(2) Structure and union type (including class type)

- Size

  Total size of members and the area for their alignment. For unions, the

overlapped area of members is not included in the size.

- Alignment

  Requires the alignment that has the largest value of the following:

- Maximum value of the alignments required by its member variables
- 4 bytes

Note The C++ compiler may add some internal data members to C++ language classes, structures, or unions, in order to implement the C++ language specification. In this case, their size and alignment may not comply with the above.

9.2.1.3 Other types

In addition to the above data types, provides the following data types.

(1) **size_t**

Corresponds with **unsigned long**.

(2) **ptrdiff_t**

Corresponds with **long**. **ptrdiff_t** is defined in the header <stddef.h> and <cstddef>.

(3) **wchar_t**

Corresponds with **int**.

(4) **bit-fields**

Specifies number of bits (including a sign bit, if any) to a member of a structure or union.

9.2.2 Type Conversion

This section explains:

- Integral promotion
- Integral conversion
- Floating-point conversion
- Complex conversion
- Floating-point and integral conversion

- Complex and integral conversion
- Complex and floating-point conversion
- Arithmetic conversion

9.2.2.1 Integral Promotion

The integer of a given type can be converted into a different type that has a wider range than the source type. Such conversion is called integral promotion.
Integral promotion is performed according to the following rules:

- If the source value can be represented as **int**, convert it into **int**.
- If it cannot be represented as **int** but as **unsigned int**, convert it into **unsigned int**.
- If it cannot be represented as **unsigned int** but as **long**, convert it into **long**.
- If it cannot be represented as **long** but as **unsigned long**, convert it into **unsigned long**.

9.2.2.2 Integral Conversion

This section explains integral conversions such as the following:

- Conversion of signed integer into unsigned integer
- Conversion of unsigned integer into signed integer

Integral conversions are performed between the integer types **char**, **short, int**, **long**, **long long**, and their **unsigned** versions.

(1) Conversion of signed integer into unsigned integer

A signed integer can be converted into the corresponding unsigned type. By this conversion, the value may be interpreted as different from the source although its bit-image is not changed.

Example:

```
#include <iostream>
main()
{
    short s = -1;
    unsigned short u;

    std::cout << s << std::endl;
    u = s;
    std::cout << u << std::endl;
}
```

"-1" and "65535" are displayed by this program. The variable "s" which has **signed short** is initialized to "-1", a negative value. The value is converted to **unsigned short** and assigned to the variable "u" by the assignment "u = s".

(2) Conversion of unsigned integer into signed integer

An unsigned integer can be converted into the corresponding signed type. If the value of the unsigned integer cannot be expressed as a signed type, the value may be interpreted incorrectly.

Example:

```
#include <iostream>
main()
{
    short s;
    unsigned short u = 65535;

    std::cout << u << std::endl;
    s = u;
    std::cout << s << std::endl;
}
```

"65535" and "-1" are displayed by this program. The variable "u" is **unsigned short** and it must be converted to a signed type to enable the assignment "s = u". However, the value "65535" is interpreted as the wrong value "-1" because it cannot be represented correctly as **signed short**.

9.2.2.3 Floating-point Conversion

A floating-point type can be converted safely into another floating-point type if the destination type has more precision than the source type. "Safely" means that there

is no loss of precision occurring during conversion. For example, conversion of **float** to **double** and conversion of **double** to **long double** are safe.

A floating-point type can also be converted into another floating-point type that has less precision than the source type. This conversion can be performed only if the source value is in a range that can be expressed by the destination type. The result is converted to the nearest value of the source value when a loss of precision occurs. If the source value is not in a range that can be expressed by the destination type, the conversion may create an undefined result.

Example:

```
#include <iostream>
main()
{
    double d = 1e+100;
    long double ld;
    float f;

    ld = d;
    std::cout << ld << std::endl;
    f = d;
    std::cout << f << std::endl;
}
```

"1e+100" and "inf" are displayed by this program. The first output means that the conversion of **double** to **long double** is performed safely. The second output means that the source value "1e+100" in **double** cannot be expressed in **float** and is converted to infinity because the maximum value that can be expressed in the float type is "3.40282347e+38".

9.2.2.4 Complex Conversion

A complex type can be converted into another complex type if the destination type has more or less precision than the source type. When a complex type is converted into another complex type, both the real and imaginary parts are converted according to the same conversion rules as floating-point conversion. See **Floating-point Conversion** for the details.

9.2.2.5 Floating-point and Integral Conversion

A floating-point type can be converted into an integer type and an integer type can be converted into a floating-point type.

(1) Conversion of floating-point into integer

When a floating-point type is converted into an integer type, the fractional part of the source value is truncated; that is, "1.3" is converted to "1" and "-1.2" is converted to "-1" for example. Rounding is not performed during conversion.

(2) Conversion of integer into floating-point

When an integer type is converted into a floating-point type, the result is exact if possible. Otherwise, the source value is converted to the nearest value that can be expressed by the destination type.

9.2.2.6 Complex and Integral Conversion

A complex type can be converted into an integer type and an integer type can be converted into a complex type.

(1) Conversion of complex into integer

When a complex type is converted into an integer type, the imaginary part of the complex value is discarded and the fractional part of the source value of the real part is truncated. For example, when a complex value has a real part of "1.3" and an imaginary part of "2.0", it is converted to "1". Rounding is not performed during conversion.

(2) Conversion of integer into complex

When an integer type is converted into a complex type, the imaginary part of the complex result value is signed zero and the real part of the complex result value is exact if possible. Otherwise, the source value is converted to the nearest value that can be expressed by the destination type.

9.2.2.7 Complex and Floating-point Conversion

A complex type can be converted into a floating-point type and a floating-point type can be converted into a complex type.

(1) Conversion of complex into floating-point

When a complex type is converted into a floating-point type, the imaginary part of the complex value is discarded and the real part of the complex value is converted into the destination type according to the same conversion rules as floating-point conversion. See **Floating-point Conversion** for the details.

(2) Conversion of floating-point into complex

When a floating-point type is converted into a complex type, the imaginary part of the complex result value is signed zero and the source value is converted into the real part of the complex type according to the same conversion rules as floating-point conversion. See **Floating-point Conversion** for the details.

9.2.2.8 Arithmetic Conversion

Many binary operators can have their operands converted and yield result types in a similar way. The purpose is to yield a common type, which is also the type of the result. This pattern is called usual arithmetic conversion, which is performed by the C++ compiler according to the following rules:

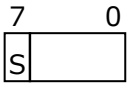
- If either operand is **long double _Complex**, the other shall be converted to **long double _Complex**.
  - Otherwise, if either operand is **long double**, the other shall be converted to **long double**.
- Otherwise, if either operand is **double _Complex**, the other shall be converted to **double _Complex**.
  - Otherwise, if either operand is **double**, the other shall be converted to **double**.
- Otherwise, if either operand is **float _Complex,** the other shall be converted to **float _Complex**.
  - Otherwise, if either operand is **float**, the other shall be converted to **float**.
- Otherwise (if neither of the operands are floating-point types nor complex types), integral promotion (see **Integral Promotion**) shall be performed on both operands as follows:
  - If either operand is **unsigned long**, the other shall be converted to **unsigned long**.
  - Otherwise, if one operand is **long** and the other is **unsigned int**, both operands shall be converted to **unsigned long**.
  - Otherwise, if either operand is **long**, the other shall be converted to **long**.
  - Otherwise, if either operand is **unsigned int**, the other shall be converted to **unsigned int**.
  - Otherwise, both operands are **int**.

9.2.3 Internal Representation of Data

9.2.3.1 Integer Types

- **signed char** (1-byte signed integer)

  SYNOPSIS

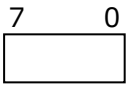
  S:Sign bit (0:positive 1:negative)

  EXPRESSIBLE VALUE

  -128 to 127 (-2^7 to 2^{7-1})

- **unsigned char** (1-byte unsigned integer)

  SYNOPSIS

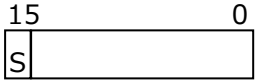
  

  EXPRESSIBLE VALUE

  0 to 255

- **short** / **short int** (2-byte signed integer)

  SYNOPSIS

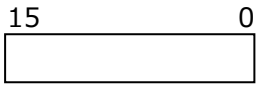
  S:Sign bit (0:positive 1:negative)

  EXPRESSIBLE VALUE

  -32768 to 32767 (-2^{15} to 2^{15-1})

- **unsigned short** / **unsigned short int** (2-byte unsigned integer)

  SYNOPSIS

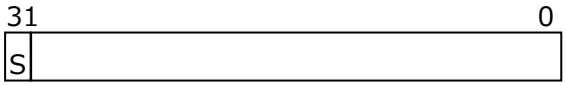
  

  EXPRESSIBLE VALUE

  0 to 65535 (0 to 2^{16-1})

- **int** (4-byte signed integer)

  SYNOPSIS

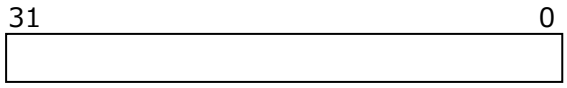
  S:Sign bit (0:positive 1:negative)

EXPRESSIBLE VALUE

-2147483648 to 2147783647 (-2^{31} to 2^{31-1})

- **unsigned int** (4-byte unsigned integer)

SYNOPSIS

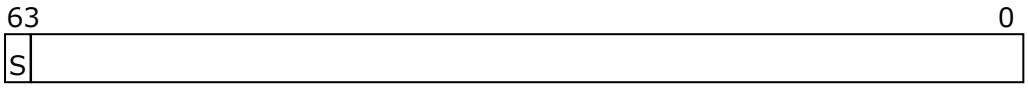


EXPRESSIBLE VALUE

0 to 4294967295 (0 to 2^{32-1})

- **long** / **long int** / **long long** / **long long int** (8-byte signed integer)

SYNOPSIS



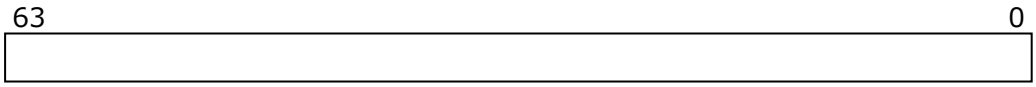
S:Sign bit (0:positive 1:negative)

EXPRESSIBLE VALUE

-9223372036854775808 to 9223372036854775807 (-2^{63} to 2^{63-1})

- **unsigned long** / **unsigned long int** / **unsigned long long** / **unsigned long long int** (8-byte unsigned integer)

SYNOPSIS



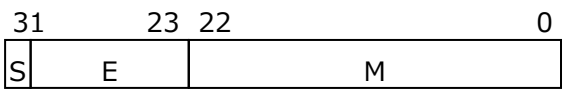
EXPRESSIBLE VALUE

0 to 18446744073709551615 (0 to 2^{64-1})

9.2.3.2 Floating-Point Types

- **float** (single-precision floating-point)

SYNOPSIS



S: Sign bit of mantissa (0:positive 1:negative)

E: Exponent (0<=E<=255)

M: Mantissa (0<=M<1)

EXPRESSIBLE VALUE

(-1)S * 2E-127 * (1.M)

Decimal value of 7 digits, with an absolute value of 0 or in the range of 10^{-38} to 10^{37}.

SPECIAL VALUE

NaN E == 255 and M != 0

Infinity E == 255 and M == 0

Signed Zero E == 0

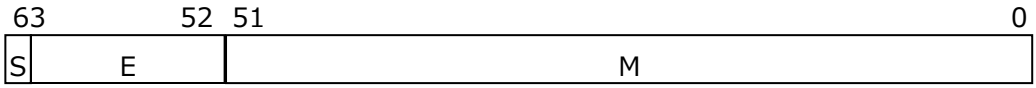
Remarks:

The compiler does not support the denormalized value (E == 0 and M != 0).

The denormalized value is handled as zero at program execution.

- **double** (double-precision floating-point)

SYNOPSIS



S: Sign bit of mantissa (0:positive 1:negative)

E: Exponent (0<=E<=2047)

M: Mantissa (0<=M<1)

EXPRESSIBLE VALUE

$(-1)^S * 2^{E-1023} * (1.M)$

Decimal value of 16 digits, with an absolute value of 0 or in the range of 10^{-308} to 10^{308}.

SPECIAL VALUE

NaN E == 2047 and M != 0

Infinity E == 2047 and M == 0

Signed Zero E == 0

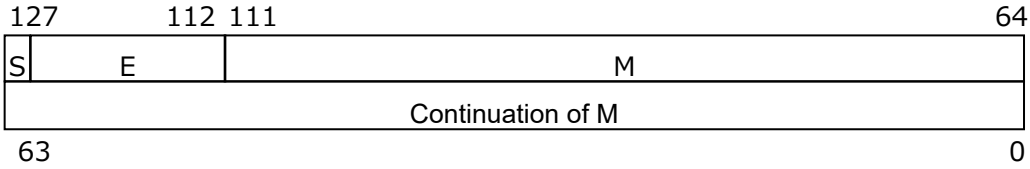
Remarks:

The compiler does not support the denormalized value (E == 0 and M != 0).

The denormalized value is handled as zero at program execution.

- **long double** (quadruple-precision floating-point)

SYNOPSIS



S: Sign bit of mantissa (0:positive 1:negative)

E: Exponent for leading digits

M: Mantissa for leading digits

EXPRESSIBLE VALUE

$(-1)^S * 2^{E-16383} * 1.M$

Decimal value of 34 digits, with an absolute value of 0 or in the range of 10^{-4932} to 10^{4932}.

SPECIAL VALUE

NaN E == 32767 and M != 0

Infinity E == 32767 and M == 0

Signed Zero E == 0

Remarks:

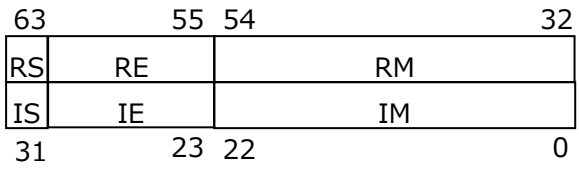
The compiler does not support the denormalized value (E == 0 and M== 0).

The denormalized value is handled as zero at program execution.

9.2.3.3 Complex Types

- **float _Complex** (single-precision complex)

SYNOPSIS



RS, IS: Sign bit of mantissa (0:positive 1:negative)

RE, IE: Exponent (0<=RE<=255, 0<=IE<=255)

RM, IM: Mantissa (0<=M<1)

EXPRESSIBLE VALUE

$(-1)^{RS} * 2^{RE-127} * (1.RM)$

$(-1)^{IS} * 2^{IE-127} * (1.IM)$

Decimal value of 7 digits, with an absolute value of 0 or in the range of 10-38

to 1037.

SPECIAL VALUE

NaN RE == 255 and RM != 0 and IE == 255 and IM != 0

Infinity RE == 255 and RM == 0 and IE == 255 and IM == 0

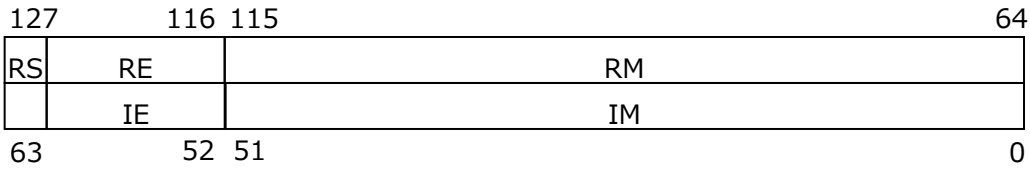
Signed Zero RE == 0 and IE == 0

Remarks:

The compiler does not support the denormalized value (RE == 0 and RM != 0, or IE == 0 and IM == 0) in the real part or the imaginary part of the complex value. The denormalized value is handled as zero at program execution.

- **double _Complex** (double-precision complex)

SYNOPSIS



RS, IS: Sign bit of mantissa (0:positive 1:negative)

RE, IE: Exponent (0<=RE<=2047, 0<=IE<=2047)

RM, IM: Mantissa

EXPRESSIBLE VALUE

$(-1)^{RS} * 2^{RE-1023} * (1.RM)$

$(-1)^{IS} * 2^{IE-1023} * (1.IM)$

Decimal value of 16 digits, with an absolute value of 0 or in the range of 10-308 to 10308.

SPECIAL VALUE

NaN RE == 2047 and RM != 0 and IE == 2047 and IM != 0

Infinity RE == 2047 and RM == 0 and IE == 2047 and IM == 0

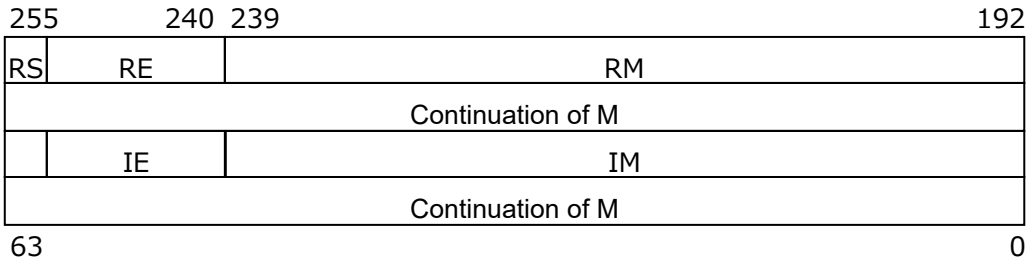
Signed Zero RE == 0 and IE == 0

Remarks:

The compiler does not support the denormalized value (RE == 0 and RM != 0, or IE == 0 and IM == 0) in the real part or the imaginary part of the complex value. The denormalized value is handled as zero at program execution.

- **long double _Complex** (quadruple-precision complex)

SYNOPSIS



RS, IS: Sign bit of mantissa (0:positive 1:negative)

RE, IE: Exponent (0<=RE<=32767, 0<=IE<=32767)

RM, IM: Mantissa

EXPRESSIBLE VALUE

$(-1)^{RS} * 2^{RE-16383} * 1.RM$

$(-1)^{IS} * 2^{IE-16383} * 1.IM$

Decimal value of 34 digits, with an absolute value of 0 or in the range of 10^{-4932} to 10^{4932}.

SPECIAL VALUE

NaN RE == 32767 and RM != 0 or IE == 32767 and IM != 0

Infinity RE == 32767 and RM == 0 or IE == 32767 and IM == 0

Signed Zero RE == 0 and IE == 0

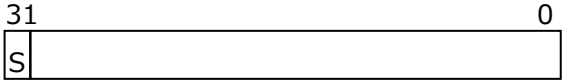
Remarks:

The compiler does not support the denormalized value ((RE == 0 and RM != 0) or (IE == 0 and IM != 0)) in the real part or the imaginary part of the complex value. The denormalized value is handled as zero at program execution.

9.2.3.4 Enumeration Type

- 4-byte

SYNOPSIS



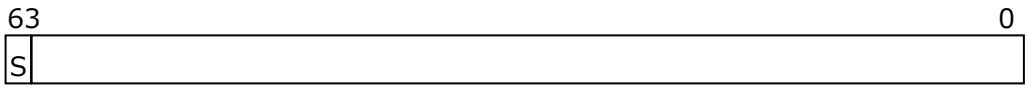
S:Sign bit (0:positive 1:negative)

EXPRESSIBLE VALUE

-2147483648 to 2147783647 (-2^{31} to 2^{31-1})

- 8-byte

SYNOPSIS



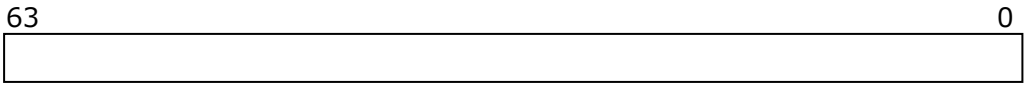
S:Sign bit (0:positive 1:negative)

EXPRESSIBLE VALUE

-9223372036854775808 to 9223372036854775807 (-2^{63} to 2^{63-1})

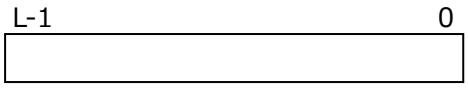
9.2.3.5 Pointer Type

SYNOPSIS



9.2.3.6 Bit Fields

SYNOPSIS



L: Length of the field

DESCRIPTION

- When the field is signed, the leftmost bit is a sign bit.
- The allocation of a field is left-oriented and tries not to go beyond the boundary of the field type. If among the allocation of several neighboring fields there is a field that exceeds its field type boundary, the allocation of that field begins from the start of the next address that has the boundary of that type.

9.2.4 Predefined Macro

All predefined macros can be shown by specifying **-dM -E**.

Example:

```
$ ncc -dM -E a.c | sort
```

The main predefined macros are as follows.

_LP64, __LP64__

Always defined as 1.

unix, __unix, __unix__

Always defined as 1.

linux, __linux, __linux__

Always defined as 1.

__ve, __ve__

Always defined as 1.

__ELF__

Always defined as 1.

__STDC__

Always defined as 1.

__STDC_HOSTED__

Always defined as 1.

__STDC_NO_ATOMICS__

Always not defined.

__NEC__

Always defined as 1.

__FAST_MATH__

Defined as 1 when **-ffast-math** is enabled; Otherwise not defined.

_FTRACE

Defined as 1 when **–ftrace** is enabled; Otherwise not defined.

__NEC_VERSION__

Defined as the value obtained by calculation using the following formula when compiler version is *X.Y.Z.*

*X**10000 + *Y**100 + *Z*

__OPTIMIZE__

Sets the optimization level n of -On which is effective at the compilation.

__VECTOR__

Defined as 1 when automatic vectorization is enabled; Otherwise not defined.

__VERSION__

Always defined as a string constant which describes the version of the compiler in use.

9.3 Inline Assembly Language

9.3.1 Basic Asm Statement

Format:

asm [**volatile**] (Assembler-instructions)

Descriptions:

Basic **asm** statement has no operands.

An optional qualifier **volatile** has no effect as all the basic **asm** statements are implicitly **volatile**.

9.3.2 Extended Asm Statement

Format:

asm [**volatile**] (*Assembler-Template*
: Output-Operands
[: Input-Operands
[: *Clobbers*]])

Assembler-Template is:

String-text of assembler instructions

Output-Operands are:

[[Symbolic-Name]] Constraint (C-variablename) , [Output-Operand] ...

Input-Operands are:

[[Symbolic-Name]] Constraint (C-expression) , [Input-Operand] ...

Clobbers are:

Register-name or “memory” , [*Clobber*] ...

Descriptions:

Extend **asm** statement can have operands. C variables can be read and written in an extended **asm** statement as its operand.

An optional qualifier volatile can be used to disable an optimization which may produce side effects while executing the **asm** statement.

Assembler-Template is a string text of assembler instructions. The compiler replaces the tokens referring to input/output operands in the *Assembler-Template* and outputs the resulting string text to the assembly codes. The Nth operand specified in the Output-Operands and the Input-Operands can be referenced with %N. %% in *Assembler-Template* is replaced with a single %.

Output-Operands are written as a comma-separated list of an output operand. An output operand has constraints and a C variable name that is modified by the assembler instructions in the Assembler-Template. An empty list is permitted.

Input-Operands are written as a comma-separated list of an input operand. An input operand has constraints and a C variable name or expressions referenced by the assembler instructions in the Assembler-Template. An empty list is permitted.

Clobbers are written as a comma-separated list of registers destroyed in assembler instructions in the Assembler-Template. “memory” is also available as a special clobber argument. A “memory” clobber informs the compiler that the assembly code generated from this **asm** statement may implicitly do memory reads/writes to items other than the operands listed in Output-Operands and Input-Operands. An empty list is permitted.

Available Constraints

Constraint	Functional specification
m	A memory operand is allowed in the form of AS or ASX.
r	Scalar register operand is allowed.
i	An integer immediate operand is allowed. This includes symbolic constants that you do not know unless values are assembled.
0, 1, 2,…9	To specify in order to allocate same register to both input operand and output operand. (Even if limited C Expression is same, the compiler does not ensure to allocate same register.) This constraint is called a matching constraint.
[Symbolic-Name]	This is a symbolic matching constraint. You can refer to %[Symbolic-Name] in the Assembler-Template.
f	Temporary scalar register operand can be allowed. This should be specified in order to store single precision floating point data in the register allocated for the operand.

Constraint priority

When you list more than one constraint (for example, "=rm0") in the constraints, the compiler selects a constraint in the order of descending priorities. The priorities are as follows.

i > r > m > matching constraint

When all of constraints in the constraints are ignored because “#” or “*” of constraint modifier characters has been specified, the compiler assumes that the r constraint is specified. There are some cautions. Please refer to notes.

Constraint Modifier Characters

Modifier Character	Functional specification
=	Means that this operand is write-only for this instruction.
+	Means that this operand is both read and written by the instruction.
&	Allocates different registers to input operand and output operand.

Modifier Character	Functional specification
%	Specifies that this operand and the next operand are interchangeable. With this specification, the compiler can exchange two operands if it is the least expensive method to satisfy all the constraints.
#	Ignores all constraints up to the next comma.
*	Ignores the next character.

Example

```
#define FIVE 5
int main(void)
{
    int in=10, out;
    asm ("ldl.sx %%s50, %1¥n¥t"
         "adds.w.zx %0, %2, %%s50¥n¥t"
         : "=r"(out)         /* Constraint "r" */
         : "m"(in), "i"(FIVE)    /* Constraint "m", "i" */
         : "%s50"
    );
    printf("out=%d¥n", out);
}
```

9.3.3 Specifying name in assembler codes

Format:

Declarator asm (Name)

Descriptions:

You can specify the name of the function or variable in the assembler code. Declarator is a declarator conforming to the C language syntax. Name is a character string of the function name or variable name in assembler codes.

9.3.4 Notes

- Alternate keywords **__asm**, **__asm__**, **__volatile**, and **__volatile__** are acceptable.
- The only Extended **asm** syntax specified to scalar register, vector register, vector mask register, vector index register in an Assembler-Template is to specify clobbered register.
- Immediate value in the form of M must be directly specified in an Assembler-Template. The operand of Extended **asm** syntax cannot be specified.

- disp specified as AS operand of "br[cf].*.[bp]" ("*" is one of "l", "w", "d" or "s") instruction must be specified with constraint character "i".
- The HM operand of lhm.* and shm.* ("*" is "b", "h", "w" or "l") instructions must be specified with constraint character either "r" or "i".
- When immediate value outside the range of -2147483648 to 2147483647 is specified as input/output operand in C-expression, the operand must be specified with constraint character either "r" or "m".
- Instruction to jump outside **asm** statement is not allowed.
- When the "number" of %number in an assembler template is not smaller than or equal to the total number of input "+" output operands, %number is not replaced and is output to the assembler file.
- When quadruple precision floating-point arithmetic instructions such as FAQ, FSQ, FMQ and FCQ instructions are used in an Assembler-Template, the extended **asm** syntax cannot be used.
- Only automatic variable, register variable or immediate value can be specified as input/output operand in C-expression.

9.4 Remarks

9.4.1 Remarks for C language

- _Atomic qualifier is not supported.
- **CX_LIMITED_RANGE** pragma is ignored.
- **FENV_ACCESS** and **FP_CONTRACT** are ignored.
- When the following C headers of the C standard library are used, specify **-pthread**.
  <stdatomic.h>, <threads.h>
- The following GCC extensions are unavailable.
  - The forward declaration of function parameters (so they can participate in variable-length array parameters).
  - GNU-style complex integral types
  - Nested functions

- Local structure with a variable-length array field. Such a field is treated (with a warning) as a zero-length array in GNU C mode, which is a useful approximation in some circumstances, but not generally equivalent to the GNU feature.
- Label as value

9.4.2 Remarks for C++ language

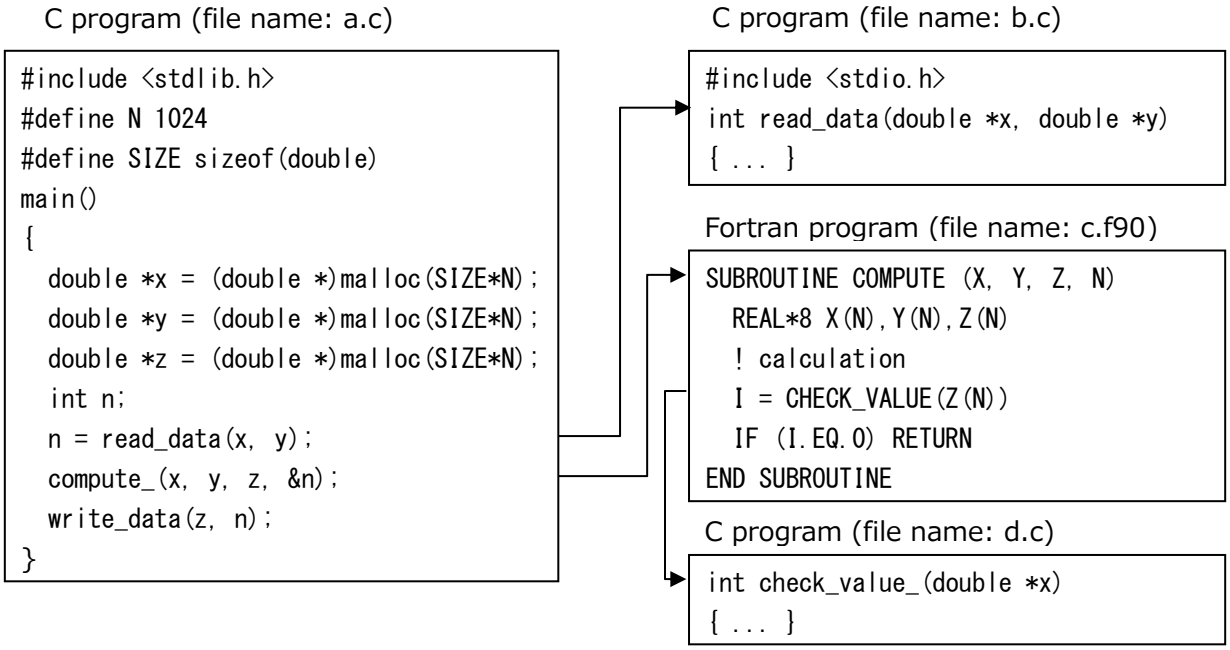
- Sized deallocation is unavailable.
- When the following C++ headers of the C++ standard library are used, specify **-pthread**.
  <atomic>, <condition_variable>, <mutex>, <thread>
- The following features which have been added and/or extended by C++17 standard are not available.
  - Class template scoped_lock
  - Class template specialization owner_less<void>
  - Template member function merge() of associative containers defined in the headers <map>, <set>, <unordered_map>, and <unordered_set>.
  - Array support to class template shared_ptr.

Chapter10 Language-Mixed Programming

Making an executable file by linking object files from different languages is called mixed language programming. This chapter describes mixed language programming techniques using C/C++ and Fortran programs.

10.1 Point of Mixed Language Programming

The following example shows how mixed language programming is used to make an executable file by linking a C program and a Fortran program.



In this example, a Fortran program is called from a C program, and a C program is called from a Fortran program. When these programs are called, the function name and procedure name coded in the program are converted into an external symbol name, and the data is shared between C and Fortran by passing arguments or return values.

The features of mixed language programming are as follows.

- C/C++ function name and Fortran procedure name correspond.
- C/C++ and Fortran data types correspond.
- Return values are passed from C/C++ to Fortran.
- Values are passed from C/C++ to Fortran by arguments.

- Executable files are created by compiling and linking.

10.2 Correspondence of C/C++ Function Name and Fortran Procedure Name

The C++ function names and Fortran procedure names in the source files are converted into external symbol names and placed in object files. Therefore, when these functions and procedures are called, they must be called by their converted external symbol names.

10.2.1 External Symbol Name of Fortran Procedure

(1) When binding labels for procedures are used:

A procedure name in a Fortran source file is converted to an external symbol name of the string same as a binding label. In other words, when a Fortran procedure has a **NAME** specifier, the procedure name is converted to the name specified to the **NAME** specifier; otherwise the procedure name is converted to lowercase.

Example:

```
SUBROUTINE SUB1(X) BIND(C, NAME="Fortran_Sub1")
...
END SUBROUTINE
SUBROUTINE SUB2(Y) BIND(C)
...
END SUBROUTINE
```

In this example, the following procedure names are converted to external symbol names.

Procedure Name		External Symbol Name
SUB1	->	Fortran_Sub1
SUB2	->	sub2

(2) When binding labels for procedures are not used:

A procedure name in a Fortran source file is converted to an external symbol name according to the following rules.

- Procedure names are converted to lowercase.
- An underscore (_) is appended to a procedure name.

Example:

```
SUBROUTINE COMPUTE (X, Y, Z, N)
REAL*8 X(N),Y(N),Z(N)
! calculation
I = CHECK_VALUE(Z(N))
IF (I.EQ.0) RETURN
END SUBROUTINE
```

In this example, the following procedure names are converted to external symbol names.

```
Procedure Name      External Symbol Name
COMPUTE         ->  compute_
CHECK_VALUE     ->  check_value_
```

10.2.2 External Symbol Name of C++ Function

The C++ compiler appends a string showing the return value and argument type to a function name in a C++ source file. This operation is called mangling a function name. By using this operation, the C++ compiler can declare functions with the same name but whose argument types differ.

Example:

```
Function Name in A Source File Mangled Name
void func(double *x)        ->  _Z4funcPd
void func(float *x)         ->  _Z4funcPf
```

Note Converting a mangled name to a name in a C++ source file is called demangling.

A C++ function called from a C function or a Fortran procedure should be declared by C linkage so that the function name is not mangled, and the C++ function can be called by the function name itself coded in the source file. In the same way, a prototype declaration of a C function or a Fortran procedure called from a C++ function should also be declared by C linkage.

Example:

```
extern "C" {
    void func(double *x);
    void func(float *x);
};
```

The linkage specification is available in C++ language only. When using a prototype declaration in C language, the linkage specification should be coded using conditional coding.

Example:

```
#ifdef __cplusplus              // __cplusplus is automatically defined
                            // by the C++ compiler.
extern "C" {
#endif
    void func(double *x);
    void func1(float *x);
#ifdef __cplusplus
};
#endif
```

10.2.3 Rules for Corresponding C/C++ Functions with Fortran Procedures

- When a Fortran procedure is called from a C function, the Fortran procedure should be called using an external symbol name of the Fortran procedure.
- A name of a C function called from a Fortran procedure should be defined by an external symbol name of the Fortran procedure.
- A C++ function called from a C function or a Fortran procedure should be declared using C linkage.
- A prototype declaration of a C function or Fortran procedure called from a C++ function should be declared using C linkage.

10.2.4 Examples of Calling

Example: Calling Fortran procedure that has the **BIND** attribute from C function.

Caller (C function)

```
extern void sub1();
void cfunc() {
    ...
    sub1();
    ...
}
```

Callee (Fortran procedure)

```
SUBROUTINE SUB1() BIND(C)
...
END SUBROUTINE SUB1
```

The Fortran procedure is declared as a prototype and called using a name that is coded in lowercase.

Example: Calling Fortran procedure that does not have the **BIND** attribute from C function.

Caller (C function)

```
extern int sub_();
void cfunc() {
    ...
    sub_();
    ...
}
```

Callee (Fortran procedure)

```
SUBROUTINE SUB
...
END SUBROUTINE SUB
```

The Fortran procedure is declared as a prototype and called using a name that is appended with an underscore (_) and coded in lowercase.

Example: Calling C function from Fortran procedure that has the **BIND** attribute.

Caller (Fortran procedure)

```
SUBROUTINE SUB
    USE, INTRINSIC :: ISO_C_BINDING
    INTERFACE
        SUBROUTINE CFUNC() BIND(C)
        END SUBROUTINE CFUNC
    END INTERFACE
    ...
    CALL CFUNC
    ...
END SUBROUTINE SUB
```

Callee (C function)

```
void cfunc() {
    ...
}
```

The C function is declared and defined using a name that is coded in lowercase, and the Fortran procedure interface is defined and called using a name that is coded in uppercase.

Example: Calling C function from Fortran procedure that does not have the **BIND** attribute.

Caller (Fortran procedure)

```
SUBROUTINE SUB
...
CALL CFUNC
...
END SUBROUTINE SUB
```

Callee (C function)

```
int cfunc_() {
    ...
}
```

The C function is declared and defined using a name that is appended with an underscore (_) and coded in lowercase.

Example: Calling Fortran procedure from C++ function.

Caller (C++ function)

```
extern "C" {
    int sub_(void);
};
void cfunc() {
    ...
    sub_();
    ...
}
```

Callee (Fortran procedure)

```
SUBROUTINE SUB
...
END SUBROUTINE SUB
```

The Fortran procedure is declared as a prototype via C linkage and called using a name that is appended with an underscore (_) and coded in lowercase.

Example: Calling C++ function from Fortran procedure.

Caller (Fortran procedure)

```
SUBROUTINE SUB
...
CALL CFUNC
...
END SUBROUTINE SUB
```

Callee (C++ function)

```
extern "C" {
    int cfunc_(void);
};
int cfunc_(void) {
    ...
}
```

The C++ function is declared and defined via C linkage using a name that is appended with an underscore (_) and coded in lowercase.

10.3 Data Types

The correspondence between Fortran data types and C/C++ data types is shown below.

10.3.1 Integer and Logical Types for Fortran

Data Type	Fortran	C/C++
Integer	**INTEGER**	**int** (*)
	INTEGER(KIND=1) **INTEGER*1**	**signed char**
	INTEGER(KIND=2) **INTEGER*2**	**short**
	INTEGER(KIND=4) **INTEGER*4**	**int**
	INTEGER(KIND=8) **INTEGER*8**	**long, long int, long long or long long int**
Logical	**LOGICAL**	**int** (*)
	LOGICAL(KIND=1)	**signed char**
	LOGICAL(KIND=2)	**short**
	LOGICAL(KIND=4)	**int**
	LOGICAL(KIND=8)	**long, long int, long long or long long int**

(*) When **-fdefault-integer**=8 is enabled: **long long int**, **long int**, **long long** or **long long int.**

10.3.2 Floating-point and Complex Types for Fortran

Data Type	Fortran	C/C++
Floating-point	**REAL**	**float** (*1)
	REAL(KIND=4) **REAL*4**	**float**
	DOUBLE PRECISION	**double** (*2)
	REAL(KIND=8) **REAL*8**	**double**
	QUADRUPLE PRECISION **REAL(KIND=16)** **REAL*16**	**long double**
Complex	**COMPLEX**	**float __complex__** (*3)
	COMPLEX(KIND=4) **COMPLEX*8**	**float __complex__**
	COMPLEX(KIND=8) **COMPLEX*16**	**double __complex__**
	COMPLEX(KIND=16) **COMPLEX*32**	**long double __complex__**

(*1) When **-fdefault-real**=8 is enabled: **double**

(*2) When **-fdefault-double**=16 is enabled: **long double**

(*3) When **-fdefault-real**=8 is enabled: **double __complex__**

10.3.3 Character Type for Fortran

Data Type	Fortran	C/C++
Character	**CHARACTER(LEN=n) ch**	**char ch**[*n*];

10.3.4 Derived Type for Fortran

(1) Description

A Fortran derived type that defined with the **BIND** attribute can associate with a C struct type.

Example:

Fortran program:

```
USE, INTRINSIC :: ISO_C_BINDING
TYPE, BIND(C) :: STR_TYPE   ! Define a derived type with the BIND attribute
    REAL(C_DOUBLE) :: S1, S2
END TYPE STR_TYPE

INTERFACE
    SUBROUTINE FUNC(X) BIND(C)
        USE, INTRINSIC :: ISO_C_BINDING
        TYPE(C_PTR) :: X
    END SUBROUTINE FUNC
END INTERFACE

TYPE(C_PTR) :: P
TYPE(STR_TYPE),TARGET :: F_STR

P=C_LOC(F_STR)          ! Get the C address of F_STR
CALL FUNC(P)                ! Call C function, and
! pass the C address of F_STR
...
```

C program:

```
struct str_type {       // Definition of structure
//  associated with STR_TYPE
    double s1, s2;
} *c_str;

void func(struct str_type **x) {
    c_str = *x;         // c_str points to F_STR
    ...
}
```

(2) Remarks

- The names of the corresponding components of the Fortran derived type and the C struct type need not be the same.
- A C struct type that contains a bit field or that contains a flexible array member cannot associate.
- A C struct type that contains a quadruple-precision real type or that contains a complex type cannot associate.

10.3.5 Pointer

A C pointer is associated with a Fortran data by using the derived type **C_PTR**.

(1) How to associate C pointer and Fortran data

When a C pointer is referred in a Fortran program, a derived type **C_PTR** is used.

Example:

Fortran program:

```
USE, INTRINSIC :: ISO_C_BINDING
    INTERFACE
    SUBROUTINE FUNC(X) BIND(C)
        USE, INTRINSIC :: ISO_C_BINDING
        TYPE(C_PTR) :: X
    END SUBROUTINE FUNC
END INTERFACE

TYPE(C_PTR) :: P
...
CALL FUNC(P)            ! Call C function
...
```

C program:

```
int *a;

void func(int **p) {
    *p = a;           // P in Fortran program points to a
}
```

(2) How to get C address

A C address of a Fortran allocated allocatable variable can be got by using the function **C_LOC** which returns a value of the **C_PTR** type.

Example:

Fortran program:

```
USE, INTRINSIC :: ISO_C_BINDING
INTEGER(C_INT),TARGET :: N
TYPE(C_PTR) :: N_ADDR
N_ADDR = C_LOC(N)       ! C_LOC(N) returns C address of "N"
```

(3) How to compare C addresses

The Fortran intrinsic procedure **C_ASSOCIATED** can compare C addresses. When its first argument and its second argument point the same area, **C_ASSOCIATED** returns ".TRUE."; otherwise returns ".FALSE.". When its second argument is omitted, **C_ASSOCIATED** returns ".FALSE." if its first argument is a C null pointer and returns ".TRUE." otherwise.

Example:

Fortran program:

```
MODULE MOD
USE, INTRINSIC :: ISO_C_BINDING
...
INTEGER(C_INT), BIND(C) :: X, Y
TYPE(C_PTR) :: P1, P2
...
END MODULE
PROGRAM MAIN
USE MOD
...
CALL FUNC(P1, P2)                          ! Call C function
IF ( C_ASSOCIATED(P1, P2) ) THEN ! Compare the memory areas of
    ...                                          ! P1 and P2
END IF
...
```

```
END
```

C program:

```
int x, y;
void func_(int **px, int **py) {
    *px = &x;           // When func() is called in Fortran program,
    *py = &y;           // P1 points x, and P2 points y
}
```

(4) How to associate C pointer and Fortran data pointer

A C pointer is associated with a Fortran data pointer by using the Fortran intrinsic procedure **C_F_POINTER**. **C_F_POINTER** associates a **C_PTR** type of its first argument with a data pointer of its second argument.

Example:

Fortran program:

```
MODULE MOD
USE, INTRINSIC :: ISO_C_BINDING
...
INTEGER(C_INT), BIND(C) :: X
TYPE(C_PTR), BIND(C) :: CP
INTEGER(C_INT), POINTER :: FP
...
END MODULE
PROGRAM MAIN
USE MOD
...
CALL FUNC(CP)               ! Call C function
CALL C_F_POINTER(CP, FP)    ! Bind C pointer CP with
...                             !  data pointer FP
END
```

C program:

```
int x;
void func_(int **px) {
    *px = &x;           // When func() is called in
}                       //  Fortran program, CP points x
```

10.3.6 Common Block for Fortran

(1) Description

A Fortran common block defined with the **BIND** attribute can be interoperable

with a C program. When the common block contains a single variable, it can associate with the C variable. When the common block contains two or more variables, it can associate with a C struct type. But, the Fortran common block and the C struct type must have the same number of members, and the members of the Fortran common block must have corresponding types with the corresponding members of the C struct type.

Example:

Fortran program:

```
USE, INTRINSIC :: ISO_C_BINDING
COMMON /COM1/ F1, F2
COMMON /COM2/ F3
REAL(C_FLOAT) :: F1, F2, F3
BIND(C) :: /COM1/, /COM2/          ! Specify the BIND attribute
...
```

C program:

```
struct { float f1, f2; } com1;
// The common block "COM1" which contains two or more
//  variables can associate with the struct "com1"
...
float com2;
// The common block "COM2" which contains single
//  variable can associate with the variable "com2"
...
```

(2) Remarks

- The names of the corresponding components of the Fortran common block and the C struct type need not be the same.
- A C struct type that contains a bit field or that contains a flexible array member cannot associate.
- A C struct type that contains a quadruple-precision real type or that contains a complex type cannot associate.

10.3.7 Notes

Complex, double-precision complex and quadruple-precision complex types for Fortran cannot correspond to single precision complex, double precision complex and quadruple precision complex types for C declared by using the *keyword* **_Complex**.

10.4 Type and Return Value of Function and Procedure

This section describes how to pass the return values between C functions and Fortran procedures. C++ functions can be regarded as C functions because C++ functions are called from C functions or Fortran procedures, or they are declared and defined using C linkage when they are called.

(1) Integer, logical, real, double-precision and quadruple-precision type Fortran procedures Please refer to "10.3 Data Types" for details of the correspondence between Fortran and C/C++.

Example: Calling double-precision type Fortran procedure.

Caller (C function):

```
extern double func_();
...
double a;
a = func_();            // Call Fortran procedure
...
```

Callee (Fortran procedure):

```
REAL(KIND=8) FUNCTION FUNC()
...
FUNC = 10.0
...
END FUNCTION FUNC
```

Example: Calling double-precision type C++ function.

Caller (Fortran procedure):

```
REAL(KIND=8) A
...
A = CFUNC()             ! Call C++ function
...
```

Callee (C++ function):

```
extern "C" {
    double cfunc_();
}
double cfunc_()
{
    double a;
    ...
    return a;
}
```

(2) Complex type functions

C/C++ can neither return nor receive a complex, double-precision complex or quadruple-precision complex type return value of Fortran.

(3) Character type functions

Two arguments are appended in order to return a value for a character type function of Fortran. The arguments are for the address and the length (in bytes) of the return value.

Example: Calling character-type Fortran procedure.

Caller (C++ function):

```
extern "C" {
     int chfunc_(char *res_p, long res_l);
}
char a[17];              // Allocate 16 bytes + 1 byte for terminating
...
chfunc_(a, 16L);         // Call Fortran procedure
a[16] = '¥0';
...
```

Callee (Fortran procedure):

```
CHARACTER*16 FUNCTION CHFUNC
CHFUNC = "THIS IS FORTRAN."
RETURN
END FUNCTION CHFUNC
```

A string data storage area is allocated in the C/C++ function. When a storage area is allocated in a C/C++ function, an extra 1 byte must be allocated for a null-terminator, because a Fortran string value is not null-terminated.

Example: Calling C function as character-type function.

Caller (Fortran procedure):

```
SUBROUTINE SUB
CHARACTER*20 CHFUNC, CH
INTEGER M
...
CH = CFUNC(M)           ! Call C function
...
END SUBROUTINE SUB
```

Callee (C function)

```
extern int cfunc_(char *a, long b, int *p);

int cfunc_(char *a, long b, int *p)
{
    strcpy(a, "THIS IS C++.");
}
```

The first argument of the Fortran procedure corresponds to the third argument of the C/C++ function.

(4) Fortran subroutine

A Fortran subroutine is the same as a C/C++ **int** type function.

10.5 Passing Arguments

10.5.1 Fortran Procedure Arguments

The arguments in a Fortran procedure that does not have the **VALUE** attribute are passed by addresses. And, the arguments in a Fortran procedure that have the **VALUE** attribute are passed by value. Therefore, when arguments are passed to a C/C++ function, the arguments are obtained as pointers by the C/C++ function. And, when the arguments are passed to a Fortran procedure, the arguments are passed as the addresses of the variables.

(1) Passing arguments to Fortran procedure that does not have the **VALUE** attribute

The arguments are passed to a Fortran procedure as the addresses of the variables. A constant value should be assigned to a variable before passing because constant values do not have storage areas.

Example:

Caller (C++ function):

```
extern "C" {
    int func_(int *i, int *j);
}
void c_func()
{
    int a, b, ret;
    ...
    b = 100;                    // Assign the constant value
                              // to a variable to pass
    ret = func_(&a, &b);        // Call Fortran procedure
    ...
}
```

Callee (Fortran function):

```
INTEGER FUNCTION FUNC(I, J)
INTEGER I, J
...
END FUNCTION FUNC
```

(2) Passing arguments to Fortran procedure that have the **VALUE** attribute

The arguments are passed to a Fortran procedure as the values of the variables. A constant value can be passed by the argument.

Example:

Caller (C++ function):

```
extern "C" {
    int func_(int i, int j);
}
void c_func()
{
    int a, ret;
    ...
    ret = func(a, 100);     // Call Fortran procedure
    ...
}
```

Callee (Fortran function):

```
INTEGER FUNCTION FUNC(I, J)
INTEGER,VALUE I, J          ! Specify the VALUE attribute
...
END FUNCTION FUNC
```

(3) Obtaining arguments from a Fortran procedure that does not have the VALUE attribute

The addresses of the arguments are received via pointer parameters.

Example:

Caller (Fortran procedure):

```
SUBROUTINE SUB
INTEGER K, I, J
...
K = C_FUNC(I, J)
...
END SUBROUTINE SUB
```

Callee (C function)

```
extern int c_func_(int *a, int *b);

int c_func_(int *a, int *b)
{
    ...
}
```

(4) Obtaining arguments from a Fortran procedure that have the **VALUE** attribute

The arguments are received by values.

Example:

Caller (Fortran procedure):

```
SUBROUTINE SUB
    INTERFACE
        INTEGER(C_INT) FUNCTION C_FUNC(A, B)
            USE, INTRINSIC :: ISO_C_BINDING
            INTEGER(C_INT), VALUE :: A, B
! Specify the VALUE attribute
END FUNCTION C_FUNC
    END INTERFACE
INTEGER I, J
...
CALL C_FUNC(I, J)
...
END SUBROUTINE SUB
```

Callee (C function):

```
extern int c_func(int a, int b);

int c_func(int a, int b)        // The arguments are received by values
{
    ...
}
```

10.5.2 Notes

10.5.2.1 Appending Arguments Implicitly

Arguments are implicitly appended to Fortran procedures as follows.

- When a called procedure is a character type Fortran function, the address where the function value is stored and the length (in bytes) of the function value are appended.
- When a procedure passes a character type argument, the length (in bytes) of the argument is appended.
- When a procedure passes a procedure name argument, the size (in bytes) of the return value from the procedure is appended. If the procedure is not a character type function, the length is 0 (zero).

Arguments are passed to procedures in the following order.

(1) Address where the return value is stored (when the called procedure is a character-type)

(2) Size of the return value (when the called procedure is a character-type)

(3) For each type of argument

The length (in bytes) of the argument for a character-type argument or the size (in bytes) of the return value for a procedure name argument is passed immediately after each argument.

10.6 Linking

10.6.1 Linking Fortran Program and C Program

When linking a C program and a Fortran program, use the Fortran compiler (nfort).

Example:

```
$ nfort -c a.f          (Compile Fortran program)
$ ncc -c b.c            (Compile C program)
$ nfort a.o b.o         (Linking by Fortran compiler)
```

10.6.2 Linking Fortran Program and C++ Program

When linking a C++ program and a Fortran program, use the Fortran compiler (nfort).When linking, the runtime library of the C++ compiler (**-cxxib**) must be specified.

Example:

```
$ nfort -c a.f                  (Compile Fortran program)
$ nc++ -c b.cpp                 (Compile C++ program)
$ nfort a.o b.o -cxxlib         (Linking by Fortran compiler)
```

10.7 Notes

When a C/C++ program and a Fortran program are linked, **stdin**, **stdout** and **stderr** must not be closed in the C/C++ program. If they are closed, execution of the Fortran program is not guaranteed.

Chapter11 Messages

11.1 Diagnostic Messages

The compiler outputs diagnostic messages that indicate the optimization status of the program to the standard error output and diagnostic message list. This section describes their formats and the main messages.

11.1.1 Diagnostic Message Format

Diagnostic messages will be output in the following format.

```
Kind (Number): Position: Message [: Hint]
```

Kind (Number):

The message kind and the number assigned to the message body will be displayed. The kinds include the following.

vec:	Vectorization information
opt:	Optimization and vectorization information
dtl:	Detailed optimization and vectorization information
inl:	Inlining information
par:	OpenMP and automatic parallelization
err:	Mainly, syntax error of OpenMP directive specification

Position:

The line number of the source code corresponding to the diagnostic message will be output. When output to standard error output, the file name including the line number is also output.

Message:

The text of the diagnostic message will be output.

Hint:

Depending on the diagnostic message, the procedure name, variable name, and array name will be output.

- When the variable name or array name is unknown, the type name may be output.
- A name of a procedure or variable generated by the compiler for optimization may be output with "$*number*" appended.

11.1.2 Message List

vec(101): Vectorized loop.

An entire loop structure is vectorized.

vec(102): Partially vectorized loop.

Part of a loop structure is vectorized.

vec(103): Unvectorized loop.

A loop is not vectorized.

vec(107): Iteration count is too small.

A loop is not vectorized because the iteration count of the loop is smaller than the threshold value for vectorizing. The threshold value can be changed by **-mvector-threshold**=*n*.

vec(108): Unvectorizable loop structure.

Loop structure does not meet vectorization conditions. This diagnostic is mainly output in the following cases.

- The loop induction variable appears in type conversion operation. It may be vectorized by **-mreplace-loop-induction**.
- The loop control expression is not an expression to compare an induction variable and a loop invariant expression.
- A logical and (&&) or a logical or (||) operation appears in the loop control expression.
- An equation operation (!= or ==) appears in the loop control expression. It may be vectorized by **-mreplace-loop-equation**.
- There are two or more branches to outside of a loop.
- There is a jump from outside of a loop. This situation appears when the loop is composed of **if** and **goto** statements.
- A work vector for partially-vectorization cannot be created. The following code shows an example that a work vector for “a[0]” is required but its type is unvectorizable and the compiler cannot prepare any work vector.

```
void func(
    int n,
    long double _Complex a[n],
    double _Complex b[n],
    double _Complex c[n],
    double _Complex d[n]
)
{
    for (int i = 0; i < n; i++) {
        a[0] = b[i] + d[i] + c[i];
        c[i] = a[0];
    }
}
```

vec(109): Vectorization obstructive statement.

A loop cannot be vectorized because a statement that makes a whole loop unvectorizable appears.

vec(110): Vectorization obstructive function reference : Function-name

A loop cannot be vectorized because a function reference that makes a whole loop or array expression unvectorizable appears.

vec(111): “novector” is specified.

A loop is not vectorized because **novector** directive is specified.

vec(112): “novwork” is specified.

A loop is not partially-vectorized because **novwork** directive is specified.

vec(113): Overhead of loop division is too large.

A loop cannot be partially-vectorized because the compiler judged the overhead due to loop division to be large and the effect of the partially-vectorization to be none.

vec(115): Internal table overflow.

A loop cannot be vectorized because an internal table used in vectorization

processing overflowed.

vec(116): Unvectorizable function reference. : Function-name

A loop cannot be vectorized because there is a function reference to an external procedure, internal procedure, module procedure, or intrinsic procedure that is not subject to vectorization.

vec(117): Unvectorizable statement.

A loop cannot be vectorized because a statement is not subject to vectorization.

vec(118): Unvectorizable data type.

A loop cannot be vectorized because a data element reference is of a type that is not subject to vectorization.

vec(119): Array is not aligned. : Variable-name

A loop cannot be vectorized because an array is not aligned on a vectorizable memory boundary.

vec(120): Unvectorizable dependency. : Variable-name

A loop cannot be vectorized because there is an unvectorizable dependency in a variable or array.

vec(121): Unvectorizable dependency.

A loop cannot be vectorized because there is an unvectorizable dependency in a variable or array.

vec(122): Dependency unknown. Unvectorizable dependency is assumed. : Variable-name

An unvectorizable dependency is assumed to exist because dependency analysis is not possible. The compiler applies vectorization with the

assumption that the dependency is not unvectorizable if **ivdep** directive is specified.

vec(124): Iteration count is assumed. Iteration count=n

The compiler assumes that the loop iteration count is *n*.

vec(126): Idiom detected. : Kind of macro

A vector macro operation is detected. The following kinds are detected.

Max/Min, List Vector, Sum, Product, Bit-op, Iteration, Search

vec(128): Fused multiply-add operation applied.

A fused-multiply-add operation is applied.

vec(129): Array is retained. : Array-name

A retain directive is applied to an array.

vec(130): Vector register is assigned.: Array-name

A vector register is assigned to an array by a **vreg** directive.

vec(131): Too many statements.

A loop cannot be vectorized because there are too many statements in a loop.

vec(132): Too many function calls.

A loop cannot be vectorized because there are too many function calls in a loop.

vec(133): Too many memory refereneces.

A loop cannot be vectorized because there are too many memory references in a loop.

vec(134): Too many branches.

A loop cannot be vectorized because there are too many branches.

vec(139): Packed loop.

A loop is vectorized by using packed-vector instructions.

vec(140): Unpacked loop. : Reason

-mvector-packed or **packed_vector** directive is specified, but any packed-vector instruction is not used in vectorization.

vec(141): “nopacked_vector” is specified.

nopacked_vector directive is applied.

vec(142): pvreg is used in vector loop.

An array which is specified by **pvreg** directive appears in a vectorized loop without packed-vector instructions.

vec(143): vreg is used in packed vector loop.

An array which is specified by **vreg** directive appears in a vectorized loop with packed-vector instructions.

vec(161): Structure assignment obstructs vectorization.

A loop cannot be vectorized because there is a large struct, union, or class assignment.

vec(163): Exception handling obstructs vectorization.

A loop cannot be vectorized because there are some expressions relaed to C++ exception handling.

vec(184): Division obstructs vectorization.

A loop cannot be vectorized because there is unvectorizable division.

vec(185): Exponentiation obstructs vectorization.

A loop cannot be vectorized because there is unvectorizable exponentiation.

opt(1011): Too large to optimize -- reduce program or loop size.

Optimization of this loop is inhibited because the program or the loop is too large. The program or the loop should be partitioned.

opt(1019): Feedback of scalar value from one loop pass to another.

A scalar variable accesses a value that is defined on another loop pass.

opt(1025): Reference to this function inhibits optimization.

Reference to this function inhibits optimization.

opt(1025): Reference to this procedure inhibits optimization.

Reference to this procedure inhibits optimization.

opt(1034): Multiple store conflict.

The same array element is defined more than once.

opt(1037): Feedback of array elements.

Same array element is referenced/defined on another loop pass.

opt(1038): Loop too complex -- optimization of this loop halted.

Optimization of this loop is halted because the loop is too complex.

opt(1056): Loop nest too deep for optimization.

Optimization of this loop is halted because nest of the loop is too deep.

opt(1057): Complicated use of variable inhibits loop optimization.

Optimization of this loop is inhibited because usage of the variable is too

complicated.

opt(1059): Unable to determine last value of scalar temporary.

Last value of the scalar temporary is unable to determine.

opt(1061): Use of scalar under different condition causes feedback.

A scalar variable is accessed under different conditions.

opt(1062): Too many data dependency problems.

Too many data dependency inhibits optimization.

opt(1082): Backward transfers inhibit loop optimization.

Optimization of this loop is inhibited because of backward transfer in the loop.

opt(1083): Last value of promoted scalar required.

A scalar variable that is changed to temporary array needs last value.

opt(1084): Branch out of the loop inhibits optimization.

Optimization of this loop is inhibited because of a branch out from the loop.

opt(1097): This statement prevents loop optimization.

This statement prevents loop optimization.

opt(1108): Reduction function suppressed -- need associative transformation.

The optimization with **-fmatrix-multiply** is suppressed due to **-fassociative-math** is disabled.

opt(1117): Indirect branch inhibits to optimization of loop.

Optimization of this loop is inhibited because of an indirect branch in the loop.

opt(1128): Branching too complex to optimize at this optimization level.

Optimization of this loop is inhibited because branchings in the loop are too complex.

opt(1130): Conditional scalar inhibits optimization of outer loop.

A conditional scalar definition inhibits optimization of outer loop.

opt(1131): Function references in iteration count inhibits optimization.

Function references in iteration count inhibits optimization.

opt(1166): Potential dependency due to pointer -- use restrict qualifier if ok.

Potential dependency due to pointer inhibits optimization. If **ivdep** directive is specified, the compiler considers the dependency to be optimizable and vectorizable.

inl(1214): Expansion routine is too big for automatic expansion.: Routine-name

The size of routine is too big and the routine cannot be inlined. It may be inlined by **-finline-max-function-size**=*n* or **-finline-max-times**=*n*.

inl(1219): Nesting level too deep for automatic expansion. : Routine-name

Nesting level of the expansion routine is too deep. It may be inlined by **-finline-max-depth**=*n*.

inl(1222): Inlined.: Routine-name

A routine is inlined.

opt(1268): Use of pointer variable inhibits optimization.

Use of pointer variable inhibits optimization.

opt(1282): This store into array inhibits optimization of outer loop.

This store into array inhibits optimization of outer loop.

opt(1285): Not enough work to justify concurrency optimization.

Concurrency optimization is inhibited because of not enough works in the loop.

opt(1298): Use of induction variable outside the loop inhibits optimization.

Optimization of this loop is inhibited because of use of induction variable outside the loop.

opt(1299): Redefinition of induction variable in loop inhibits optimization.

Optimization of this loop is inhibited because of redefinition of induction variable in the loop.

opt(1315): Iterations peeled from loop in order to avoid dependence.

To eliminate unvectorizable dependency, forward/backward expansion of the loop is performed.

opt(1339): User parallel directives inhibits to optimization.

Optimization is inhibited because of user parallel directive specifications.

opt(1376): User function reference inhibits optimization.

Optimization is inhibited because of user function reference.

opt(1377): Must synchronize to preserve order of accesses.

Synchronization is needed to preserve order of accesses.

opt(1378): Many synchronizations needed.

Too many synchronizations inhibits concurrency.

opt(1380): User function references not ok without "cncall".

Concurrency optimization is inhibited because of user function reference. It may be optimized if **cncall** directive is specified.

inl(1388): Inlining inhibited: OpenMP or parallel directive.

Parallelization control option exists in a candidate for inlining.

opt(1395): Inner loop stripped and strip loop moved outside outer loop.

Outer loop strip mining is performed.

opt(1408): Loop interchanged.

Outer loop is interchanged with inner loop.

opt(1409): Alternate code is generated.

Alternate code is generated.

opt(1589): Outer loop moved inside inner loop(s).

Outer loop is switched with inner loop.

opt(1590): Inner loop moved outside outer loop(s).

Inner loop switched with outer loop.

opt(1592): Outer loop unrolled inside inner loop.

Outer loop unrolling is performed.

opt(1593): Loop nest collapsed into one loop.

Nested loop collapsing is performed.

opt(1772): Loop nest fused with following nest(s).

Loop fusion with following loop is performed.

opt(1800): Idiom detected (matrix multiply).

Replace matrix multiply loop with vectorized library call.

11.2 Runtime Error Message

The compiler runt-time routine outputs error messages that indicate the program error to the standard error output. This section describes their main messages.

C++ runtime abort: terminate() called by the exception handling mechanism.

terminate() function was called by the exception handling mechanism.

C++ runtime abort: returned from a user-defined terminate() routine.

A user-defined terminate() function returned.

C++ runtime abort: internal error: static object marked for destruction more than once.

Static object marked for destruction was destroyed more than once.

C++ runtime abort: a pure virtual function was called.

A pure virtual function was called.

C++ runtime abort: invalid dynamic cast.

dynamic_cast to subobject was invalid.

C++ runtime abort: invalid typeid operation.

typeid operation was invalid.

C++ runtime abort: freeing array not allocated by an array new operation.

An array that was not allocated by new operator was freed.

C++ runtime abort: terminate() called itself recursively.

terminate() function was called recursively.

C++ runtime abort: a deleted virtual function was called.

A delete virtual function was called.

Compatibility Error: veos (older than v2.6.0) and ve_exec (vVEOS-verision) are not compatible

veos version is old, so it does not have compatibility with ve_exec. If VE program is running on a container, please install the latest veos packages to the host machine.

Compatibility Error: veos (vVEOS-version-A) and ve_exec (vVEOS-verision-B) are not compatible

veos version is old, so it does not have compatibility with ve_exec. If VE program is running on a container, please install the latest veos packages to the host machine.

Failed to load EXEC DATA (fixed): Error Message

Failed to load the data of exec file. VE memory shortage may be occured. If there is executing VE process, please terminate it or reduce the size of data. You can refer to the VE memory capacity and VE memory usage with "/opt/nec/ve/bin/free -h".

Failed to load EXEC DATA (fixed, fileback): Error Message

Failed to load the data of exec file. VE memory shortage may be occured. If there is executing VE process, please terminate it or reduce the size of data. You can refer to the VE memory capacity and VE memory usage with "/opt/nec/ve/bin/free -h".

Unable to grow stack

Size of stack is not enough. As following example, please increase the limit of the available stack size with the environment variable **VE_LIMIT_OPT**.

```
export VE_LIMIT_OPT="-s 8192"
```

You can refer to the current limit of stack size by ve_exec command with "—show-limit" as the argument.

```
$ ve_exec --show-limit
core file size          (blocks, -c) 0                    0
data seg size           (kbytes, -d) unlimited            unlimited
pending signals                 (-i) 379523               379523
max memory size         (kbytes, -m) unlimited            unlimited
stack size              (kbytes, -s) unlimited            unlimited  <--
cpu time               (seconds, -t) unlimited            unlimited
virtual memory          (kbytes, -v) unlimited            unlimited
```

VE Node node-number is UNAVAILABLE

The VE card whose number is *node-number* is fault occurs. Please use other VE node to execute job.

Chapter12 Troubleshooting

12.1 Troubleshooting for compilation

The error "Fatal: License: Unknown host." occurs.

There is a possibility that the problem that the machine can't access a license server occurs to the time of license check of a compiler. Please refer to the FAQ indicated on a following page of HPC software license issue.

https://www.hpc-license.nec.com/aurora/

When not solving it, please contact us from the said page.

The error "Syntax error" occurs at a compiler directive.

Please confirm whether the spelling of compiler directive and the how to use aren't wrong. When it's an error to compiler directive of a SX compiler, please change to it of a VE compiler by a compiler directive line change tool.

Please refer to "Appendix C Compiler Directive Conversion Tool" to confirm the usage of the tool.

The error "Error: Invalid suffix" occurs.

There is a possibility that binutils-ve package is old. Please confirm whether binutils-ve package is the latest edition.

When using a header file and a library, I want to confirm the directory to which a compiler and a linker refer.

Please refer to "1.6 Searching files specified by #include directive" and "1.7 Searching Libraries".

The error "undefined reference to 'ftrace_region_begin' / 'ftrace_region_end'" occurs at linking.

The FTRACE function is used. Specify **-ftrace** at linking.

Please refer to "PROGINF/FTRACE User's guide" about the FTRACE function.

```
$ ncc a.o b.o -ftrace
```

The error "undefined reference to '__vthr$_barrier'" occurs at linking.

Please specify **-mparallel** or **-fopenmp** at linking.

The error "undefined reference to '__vthr$_pcall_va'" occurs at linking.

Please specify **-mparallel** or **-fopenmp** at linking.

The error "cannot find -lveproginf" and "cannot find -lveperfcnt" occurs at linking.

Please install nec-veperf package.

I want to confirm whether they are executable file for VE.

Please execute "/opt/nec/ve/bin/nreadelf -h" that specified the executable file as an argument of command. When "NEC VE architecture" is output in the line of "Machine:", it show that a file is an executable file for VE.

```
$ /opt/nec/ve/bin/nreadelf -h a.out
ELF Header:
Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              EXEC (Executable file)
Machine:                           NEC VE architecture
(...)
```

When linking OpenMP and automatic parallelized program, which of -fopenmp and -mparallel should I specify?

Please specify either **-fopenmp** or **-mparallel**.

```
$ ncc -c -mparallel a.c
$ ncc -c -fopenmp b.c
$ ncc -fopenmp a.o b.o
```

When specifying -ftrace, execution time becomes so long.

It becomes long because the routine for getting performance information is

executed. It is called at entrance/exit of functions and user specified region. Please specify **-ftrace** to only the source file which includes routine which performance information is required.

Even if setting value bigger than 8 to OMP_NUM_THREADS, threads more than 8 is not generated.

8 threads are the upper limit because the number of cores of VE is 8.

I want to know the name of predefined macro and the value.

Please refer to “9.2.4 Predefined Macro”.

The following error occur when linking C++ program.

```
/opt/nec/ve/bin/nld: __curr_eh_stack_entry: TLS reference in /tmp/nccwvkaaa.o
mismatches non-TLS reference in /opt/nec/ve/ncc/2.x.x/lib/libnc++.a(iostream.o)
/opt/nec/ve/ncc/2.x.x/lib/libnc++.a: error adding symbols: Bad value
```

Please recompile the program by the compiler of version 2.2.1 or later.

When compiling a program which code size is large, the compiler aborts by SIGSEGV.

The stack size needed by the compiler may exceed upper limit of the setting. It may solve to extend the upper limit of it. It can be confirm and setting to invoke “ulimit -s” as follows. Please increase the upper limit of stack size and recompile the program.

```
$ ulimit -s          (Check the current limit)
8192
$ ulimit -s 16384    (Change the limit)
```

The compiler aborts by SIGKILL.

The memory of the machine may be exhausted. The memory used amount can be somewhat reduced to compile with **-O0** or **-O1**.

I want to link Fortran program and C/C++ program.

Please refer to “10.6 Linking”.

I want to change the options of SX series to it of Vector Engine.

Please change it to refer to “Appendix B SX Compatibility”.

I want to change the compiler directives of SX series to it of Vector Engine.

Please use the “Compiler Directive Conversion Tool” or change by hand by confirming “Appendix B SX Compatibility”. Please refer to “Appendix C Compiler Directive Conversion Tool” about the tool.

The variable or routine name which name is “$” and number as ‘$1’ is displayed in diagnostic message. What is it?

It is created by compiler to do vectorization and parallelization.

The type name as “DOUBLE” or “float” is displayed instead of variable name in diagnostic message. What is it?

It is unnamed variable created by compiler to do vectorization and parallelization. It is displayed type name because it has no name.

A compiler option which is not specified in command line is enabled.

A compiler option may be specified in option file. Please refer to “1.5 Specifying Compiler Options” to confirm details of option file.

I want to confirm version of the compiler.

Please compile with **--version**.

12.2 Troubleshooting for execution

The error “Node 'N' is Offline” occur at execution.

The state of VE node of number N is OFFLINE. Please make it ONLINE.

The example which make VE node of number 0 ONLINE state is as follows.

```
# /opt/nec/ve/bin/vecmd -N 0 state set on
...
Result: Success
# /opt/nec/ve/bin/vecmd state get
...
------------------------------------------------------------------------
VE0 [03:00.0] [ ONLINE ] Last Modif:2017/11/29 10:18:00
------------------------------------------------------------------------
Result: Success
```

I want to confirm the used node at execution.

Please execute the command /opt/nec/ve/bin/ps. The command ps outputs snapshot of executing processes by VE node. In the following example, it can be confirmed that the program named “a.out” is executing on VE node of number 2.

```
/opt/nec/ve/bin/ps -a
VE Node: 3
  PID TTY          TIME CMD
VE Node: 1
  PID TTY          TIME CMD
VE Node: 2
  PID TTY          TIME CMD
50727 pts/1    00:01:36 a.out
VE Node: 0
  PID TTY          TIME CMD
```

The error ”./a.out: error while loading shared libraries: libncc.so.2: cannot open shared object file: No such file or directory” is output at execution.

Please install the package “nec-nc++-shared” and “nec-nc++-shared-inst”. Please follow the instructions described in the "Installation Guide".

The error which a dynamic link library is not found occurs at execution.

Please set the directory which dynamic link library is put to the environment variable **VE_LD_LIBRARY_PATH**. Please refer to “2.2 Environment Variables Referenced During Execution”.

I want to confirm which line of source file corresponds to an exception occurrence point.

It can be check by traceback information. Please refer to “1.8.3 Using Traceback Information” to check process of it.

The exception occurrence point which output by traceback information is incorrect.

The exception occurrence point output by traceback information can be incorrect by the advance control of HW. The advance control can be stopped to set the environment variable **ADVANCEOFF**=YES. An execution time may increase substantially to stop the advance control. Please take care it.

```
$ export VE_ADVANCEOFF=YES
```

I want to confirm whether use uninitialized local variable or not.

It may be checked by detecting an exception to compile with **-minit-stack**=snan and execute for double type variables. For float type variables, specify snanf instead of snan. This approach can be used only if the variable is floating-point type.

I want to avoid abnormal termination caused by reference of uninitialized variable.

It may avoid by initializing the area to zero to compile with **-minit-stack**=zero and execute. Correction of a program is recommended to resolve a potential problem.

A program which uses automatic parallelization and/or OpenMP is abnormally terminated by "Unable to grow stack" or SIGSEGV at execution.

It may occur because the amount of stack usage exceeds the limit. Please increase the limit of stack size.

- The limit of stack size can be increased by setting the environment variable **OMP_STACKSIZE**.

```
$ export OMP_STACKSIZE=2G
```

I want to confirm how many thread was used at execution.

It can be confirmed to check “Max Active Threads” in PROGINF. “Max Active Threads” is output to stderr at termination when setting the environment variable **VE_PROGINF**=DETAIL. Please refer to “PROGINF/FTRACE user’s Guide” to confirm usage of PROGINF.

In the following example, it can be confirmed that 4 thread was used because “Max Active Threads” is 4.

```
******** Program Information ********
(...)
Power Throttling (sec)       :        0.000000
Thermal Throttling (sec)         :        0.000000
Max Active Threads           :        4
Available CPU Cores          :        8
Average CPU Cores Used       :        3.323850
Memory Size Used (MB)        :        7884.000000
Start Time (date)            :        Mon Feb 19 04:43:34 2018 JST
End Time (date)              :        Mon Feb 19 04:44:08 2018 JST
```

When the threads for automatic or OpenMP parallelized program execution are created or destroyed?

By default, the threads are created at the start of execution and destroyed at termination. The number of threads are the specified value by the environment variable **OMP_NUM_THREADS** or **VE_OMP_NUM_THREADS**. If it is not specified, the number is the same as the number of available VE cores.

Please refer to “7.3.2 Thread Creation and Destroy” for details.

The bus errors occur when promoting vectorization.

It may occur because vector load/store for 8 bytes elements is executed for the array aligned in 4 bytes. In the following example, the **float** type (aligned in 4 bytes) arrays “a” and “b” are which passed as arguments are casted to **uint64_t**. Therefore, it is vector load/store for 8 bytes elements.

```
void func1(){
    float a[511],b[511];
    ...
    func2(a,b);
}

void func2( void* a, void* b ){
    for(int i=0; i<255; ++i){ //!!!<---vectorized loop
        ((uint64_t*)b)[i] = ((uint64_t*)a)[i];
    }
}
```

Please align them in 8 bytes as follows or specify the **novector** directive to the loop to stop vectorization.

```
float a[511] __attribute__((aligned(8)));
float b[511] __attribute__((aligned(8)));
```

12.3 Troubleshooting for tuning

I want to confirm which optimization was applied to a program.

Please refer to output diagnostics and the format list when compiling.

The diagnostics list is output when the compiler option **-report-diagnostics,** and the format list is output when the compiler option **-report-format** is specified. For details, refer to “Chapter8 Compiler Listing”.

The performance decreases, though vectorization was promoted.

The performance decreases by an overhead of vectorization of the few iteration loop.

Please specify the **novector** directive to such loop to stop vectorization.

When automatic or OpenMP parallelized program is executed, the values displayed in the same item of PROGINF and FTRACE are different.

The number of operations for the spin-waiting of the thread created before main program starts is added in PROGINF, but not in FTRACE.

When using the $omp parallel num_threads (4) and executing with the environment variable OMP_NUM_THREADS =4 or OMP_NUM_THREADS=5, the execution time with OMP_NUM_THREADS=5 is a longer than with OMP_NUM_THREADS=4. Even though there are more parallel numbers.

When the value passed with the num_threads clause is different from the value specified with the environment variable OMP_NUM_THREADS, the execution time increases due to thread regeneration.

Threads are automatically generated before the main program starts. The number of threads is determined by the the environment variable OMP_NUM_THREADS. When the number of threads changes in the program with the function omp_set_thread_num() or num_threads clause in OpenMP, the threads generated before the main program starts is freed and the new threads are regenerated.

12.4 Troubleshooting for installation

I want to check if the installation is correct.

Please specify the **--version** option to check the version. If the displayed version number is the same as the installed property, it has been installed correctly. The version number is output to *X.X.X* in the following example.

```
$ /opt/nec/ve/bin/ncc --version
ncc (NCC) X.X.X (Build 14:10:47 Apr 23 2020)
Copyright (C) 2018,2020 NEC Corporation.
```

I want to install an older version of the compiler.

Please refer to “A.1.1 Installation of a Specific Version of the Compilers” in the SX-Aurora TSUBASA Installation Guide to install old versions of the compiler.

I want to use an older version of the compiler.

Please invoke /opt/nec/ve/bin/nfort-*X.X.X*, ncc-*X.X.X*, or nc++-*X.X.X* (*X.X.X* is the version number of the compiler) at compilation.

For detail, refer to "1.2 Usage of the Compiler".

I want to start an older version of compiler by default.

The substance of each version of ncc/nc++/nfort commands are installed as follows.

X.X.X is the version number of the compiler.

```
/opt/nec/ve/ncc/X.X.X/bin/ncc
/opt/nec/ve/ncc/X.X.X/bin/nc++
/opt/nec/ve/nfort/X.X.X/bin/nfort
```

Set the **bin** directory of the version you want to invoke by default to the command search path (environment variable **PATH**).

Chapter13 Notice

1. The version 2.0.0 or later is not compatible with the version 1.*X*.*X*. Therefore, an object file compiled by version 2.0.0 or later cannot be linked with an object file compiled by version 1.*X*.*X*.
2. Runtime library is also provided as shared library in version 2.2.2 or later. Therefore, please re-compile and re-build the shared library by version 2.2.2 or later when they were compiled by version 2.1.2 or earlier.
3. When the following error occurs at compiling of C++ program, please re-compile the source file by version 2.2.2 or later.

```
/opt/nec/ve/bin/nld: __curr_eh_stack_entry: TLS reference in /tmp/nccwvkaaa.o misma
tches non-TLS reference in /opt/nec/ve/ncc/2.2.2/lib/libnc++.a(iostream.o)
/opt/nec/ve/ncc/2.2.2/lib/libnc++.a: error adding symbols: Bad value
```

4. The dynamic linker included in glibc-ve package version 2.21-4 or later is needed to execute the executable file compiled by version 2.2.2 or later. Confirm the version of glibc-ve package if an error occurs at execution.

```
$ rpm -q glibc-ve
glibc-ve-2.21-4.el7.x86_64
```

5. The execution performance of version 2.2.2 or later may fall compared with version 2.1.2 or earlier by overhead of dynamic-link process, because the compiler links a shared library at default. It can be avoided by the compilation by **-static** or **-static-nec**.

   Notes:

   When executing the executable file compiled with **-static** or **-static-nec** option, the execution may be failed rarely. For example a result is wrong, and program aborts and so on.

Appendix A Configuration file

A.1 Overview

The configuration file can be used in order to override the defaults which the compiler uses. To use the configuration file, use **-cf**=*conf* compiler option.

The syntax of configuration file is as follow:

keyword : *value*

The following table shows currently available *keywords*.

Keyword	Description
veroot	The root directory of the VE component (default: /opt/nec/ve)
system	The root directory of the compiler component (default: /opt/nec/ve/ncc/version)
as	The path of assembler command (default: <veroot>/bin/nas)
ccom	The path of C/C++ compiler (default: <system>/libexec/ccom)
ld	The path of linker command (default: <veroot>/bin/nld)
cc_pre_options	The Compiler options. The options are specified in the follow order.
cc_post_options	<cc_pre_options> <user-specified-options> <cc_post_options>
as_pre_options	The Assembler options. The options are specified in the follow order.
as_post_options	<as_pre_options> <user-specified-options> <as_post_options>
ld_pre_options	The Linker options. The options are specified in the follow order.
ld_post_options	<ld_pre_options> <user-specified-options> <ld_post_options>
startfile	The start up file.
endfile	The start up file. The file specified as an end of a linker options.

A.2 Format

- A keyword and the value are separated by the colon.
- When a keyword is not set, it set the default value.
- A blank can be specified around the separator colon.
- When '¥' is specified as an end of a line, the value can be specified continuous in

the next line.

Example:

```
cc_pre_options:  -I /tmp ¥
-I /tmp2
```

- When specifying two or more the same keyword, the last keyword becomes effective.

A.3 Example

- Change the root directory of VE component and compiler component.
  A configuration file is made and set the value to 'veroot' and 'system'.

```
veroot: /foo/ve
system: /foo/ve/ncc/X.X.X
```

- When the configuration file is specified by **-cf** option. The configuration file name is ve.conf here.

```
$ ncc -cf=ve.conf test.c
```

- Change the using compiler.
  Only the used compiler is changed. Set the value to the "ccom" when only the used compiler is changed.

```
ccom: /foo/ve/ncc/X.X.X/libexec/ccom
```

  When the configuration file is specified by **-cf** option. An assembler, a linker and so on can also be changed in the same way.

Appendix B SX Compatibility

This appendix describes the correspondence tables of major compiler options, compiler directives, and environment variables referred at the execution between SX compilers and compilers for the Vector Engine.

B.1 Compiler Options

B.1.1 Overall Options

SX Compiler	Vector Engine Compiler
-Caopt	**-O4**
-Chopt	**-O3**
-Cvopt	**-O2**
-Csopt	**-O2 –mno-vector**
-Cvsafe	**-O1**
-Cssafe	**-O1 –mno-vector**
-Cnoopt	**-O0**
-Cdebug	**-O0 -g**
-S	**-S**
-NS	none
-V Note: Continue the compilation process.	**--version** Note: Display the version and exit.
-NV	none
-c	**-c**
-Nc	none
-cf *string*	**-cf=*string***
-clear	**-clear**
-continst | -Ncontinst	none
-dir { opt | noopt }	none

SX Compiler	Vector Engine Compiler
-f03lib	none
-f90lib [{ dw | dW | ew | eW }]	none
-o *file-name*	**-o *file-name***
-prelink	none
-size_t32	none
-size_t64	none Note: Always effective.
-syntax	**-fsyntax-only**
-Nsyntax	**-fno-syntax-only**
-to *directory-name*	none
-verbose	**-v**
-Nverbose	none

B.1.2 Vector/Scalar Optimization Options

SX Compiler	Vector Engine Compiler
-Ochg	**-fassociative-math** or **-faggressive-associative-math**
-Onochg	**-fno-associative-math**
-Odiv	**-freciprocal-math**
-Onodiv	**-fno-reciprocal-math**
-Oextendreorder	**-msched-interblock**
-Oignore_volatile	**-fignore-volatile**
-Onoignore_volatile	**-fno-ignore-volatile**
-Omove	**-fmove-loop-invariants-unsafe**
-Onomovediv	**-fmove-loop-invariants**
-Onomove	**-fno-move-loop-invariants**
-Ooverlap	**-fnamed-alias**
-Onooverlap	**-fnamed-noalias**

SX Compiler	Vector Engine Compiler
-Orestrict=arg	**-fargument-noalias**
-Orestrict=this	**-fthis-pointer-noalias**
-Orestrict=type	**-fstrict-aliasing**
-Orestrict=no	**-fargument-alias** **-fthis-pointer-alias** **-fno-strict-aliasing**
-Osafe_longjmp	none
-Onosafe_longjmp	none
-Ounroll	**-floop-unroll**
-Ounroll=*nlevel*	**-floop-unroll** **-floop-unroll-max-times**=*n* Note: Specify two at the same time.
-Onounroll	**-fno-loop-unroll**
-alias { pointer | nopointer }	none
-alias { type | notype }	none
-alias { variable | novariable }	none
-dir { vec | novec }	none
-ipa	**-fipa**
-Nipa	**-fno-ipa**
-math,scalar	**-ffast-math**
-math,vector	none
-math,nofast=*function-name*	none
-math { inline | noinline }	none
-math,round=tonearest	none
-math,round=towardzero	none
-math,round=upward	none
-math,round=downward	none
-math,strict_prototype	none
-math,nostrict_prototype	none

SX Compiler	Vector Engine Compiler
-Nmath	none
-pvctl,altcode	**-mvector-dependency-test** **-mvector-loop-count-test** **-mvector-shortloop-reduction** Note: Specify three at the same time.
-pvctl,altcode=dep	**-mvector-dependency-test**
-pvctl,altcode=nodep	**-mno-vector-dependency-test**
-pvctl,altcode=loopcnt	**-mvector-loop-count-test**
-pvctl,altcode=noloopcnt	**-mno-vector-loop-count-test**
-pvctl,altcode=shortloop	**-mvector-shortloop-reduction**
-pvctl,altcode=noshortloop	**-mno-vector-shortloop-reduction**
-pvctl,noaltcode	**-mno-vector-dependency-test** **-mno-vector-loop-count-test** **-mno-vector-shortloop-reduction** Note: Specify three at the same time.
-pvctl,assoc	**-fassociative-math**
-pvctl,noassoc	**-fno-associative-math**
-pvctl { assume | noassume }	none
-pvctl,collapse	**-floop-collapse**
-pvctl,nocollapse	**-fno-loop-collapse**
-pvctl { compress | nocompress }	none
-pvctl { conflict | noconflict }	none
-pvctl { delinearize | nodelinearize }	none
-pvctl,divloop	none
-pvctl,nodivloop	**-mwork-vector-kind**=none
-pvctl,expand=*n*	**-floop-unroll-complete**=*n*
-pvctl,noexpand	**-fno-loop-unroll-complete**
-pvctl,listvec	**-mlist-vector**
-pvctl,nolistvec	**-mno-list-vector**
-pvctl,loopchg	**-floop-interchange**

SX Compiler	Vector Engine Compiler
-pvctl,noloopchg	**-fno-loop-interchange**
-pvctl,loopcnt=*n*	**-floop-count**=*n*
-pvctl,loop_eq	**-freplace-loop-equation**
-pvctl,noloop_eq	**-fno-replace-loop-equation**
-pvctl,lstval	**-floop-last-value**
-pvctl,matmul	**-fmatrix-multiply**
-pvctl,nomatmul	**-fno-matrix-multiply**
-pvctl { neighbors | noneighbors }	none
-pvctl,nodep	**-fivdep**
-pvct,on_adb	none
-pvctl,outerunroll=*n*	**-fouterloop-unroll** **-fouterloop-unroll-max-times**=*n* Note: Specify two at the same time.
-pvctl,outerunroll_lim=*n*	none
-pvctl,replace_induction	none
-pvctl,noreplace_induction	none
-pvctl,split	**-floop-split**
-pvctl,nosplit	**-fno-loop-split**
-pvctl { vchg | novchg }	none
-pvctl,vecthreshold=*n*	**-mvector-threshold**=*n*
-pvctl,verrchk	**-mvector-intrinsic-check**
-pvctl,noverrchk	**-mno-vector-intrinsic-check**
-pvctl { vlchk | novlchk }	none
-pvctl,vregs=*n*	none
-pvctl,vwork={hybrid|stack|static}	none
-pvctl,vworksz=*n*	none
-struct,loop=*n*	none
-v	**-mvector**
-Nv	**-mno-vector**

SX Compiler	Vector Engine Compiler
-xint	**-mno-vector-iteration**
-Nxint	**-mvector-iteration**

B.1.3 Inlining Options

SX Compiler	Vector Engine Compiler
-dir { inline | noinline }	none
-pi,auto	**-finline-functions**
-pi,copy_arg	**-finline-copy-arguments**
-pi,nocopy_arg	**-fno-inline-copy-arguments**
-pi,directory=*directory-name*	none
-pi,file=*file-name*	none
-pi,func_size=*n*	none
-pi,inline	**-finline**
-pi,noinline	**-fno-inline**
-pi,max_depth=*n*	**-finline-max-depth**=*n*
-pi,max_size=*n*	**-finline-max-function-size**=*n*
-pi,search_all	none
-pi,times=*n*	**-finline-max-times**=*n*

B.1.4 Parallelization Options

SX Compiler	Vector Engine Compiler
-dir { par | nopar }	none
-Pauto	**-mparallel**
-Pmulti	none
-Popenmp	**-fopenmp**
-Pstack	none

SX Compiler	Vector Engine Compiler
-pvctl,for[=*n***]**	none Note: Parallelization schedule can be controlled by **-mschedule-static** etc.
-pvctl,by=*m*	none Note: Parallelization schedule can be controlled by **-mschedule-static** etc.
-pvctl,inner	**-mparallel-innerloop**
-pvctl,noinner	**-mno-parallel-innerloop**
-pvctl,outerstrip	**-mparallel-outerloop-strip-mine**
-pvctl,noouterstrip	**-mno-parallel-outerloop-strip-mine**
-pvctl,parcase	**-mparallel-sections**
-pvctl,noparcase	**-mno-parallel-sections**
-pvctl,parthreshold=*n*	**-mparallel-threshold**=*n*
-pvctl,noparthreshold	**-mno-parallel-threshold**
-pvctl,res={ whole | parunit | no }	none
-reserve=*n*	none

B.1.5 Code Generation Options

SX Compiler	Vector Engine Compiler
-mask { nosetall | setall | setmain }	none
-mask { flovf | flunf | fxovf | inv | inexact | zdiv }	none Note: It can be controlled by the environment variable **VE_FPE_ENABLE**.
-stkchk | -Nstkchk	none
-sx9 | -sxace	none

B.1.6 Language Options

C++/SX Compiler	Vector Engine Compiler
-Xa	none
-Xc	none

C++/SX Compiler	Vector Engine Compiler
-Xkr	none
-Xp	none
-Xs	none
-K { align8 | noalign8 }	none
-K { complex | nocomplex }	none
-Kcompound_literals	none
-Knocompound_literals	none
-Kconst_string_literals	none
-Knoconst_string_literals	none
-K { designators | nodesignators }	none
-Kexceptions	**-fexceptions** Note: Enabled by default.
-Knoexceptions Note: Enabled by default.	**-fno-exceptions**
-K { gcc | nogcc }	**-std=*keyword***
-Kgnu89_inline	**-fgnu89-inline**
-Knognu89_inline	none
-Kmultibyte_chars	none
-Knomultibyte_chars	none
-Knew_for_init	**-ffor-scope**
-Kold_for_init	**-fno-for-scope**
-Knonstd_gnu_keywords	none
-Knononstd_gnu_keywords	none
-K { nullptr | nonullptr }	none
-Kopenmp_fatal	none
-Kopenmp_warning	none
-Krestrict	**-frestrict**
-Knorestrict	**-fno-restrict**
-Kstd=*keyword*	**-std**=*keyword*

C++/SX Compiler	Vector Engine Compiler
-Ktrigraphs Note: Enabled by default.	**-trigraphs**
-Knotrigraphs	none Note: Enabled by default.
-Kunsigned_char Note: Enabled by default.	**-funsigned-char**
-Ksigned_char	**-fsigned-char** Note: Enabled by default.
-K { using_std | nousing_std }	none
-Kvariadic_templates	none
-Knovariadic_templates	none
-K { vla | novla }	none
-T { auto | noauto }	none
-T { none | all | used | local }	none
-Timplicit_include	**-fimplicit-include**

B.1.7 Performance Measurement Options

SX Compiler	Vector Engine Compiler
-acct	**-proginf**
-Nacct	**-no-proginf**
-ftrace	**-ftrace**
-ftrace { simple | demangled }	none
-Nftrace	**-no-ftrace**
-p	**-p**
-Np	none

B.1.8 Debugging Options

SX Compiler	Vector Engine Compiler
-dir { debug | nodebug }	none

SX Compiler	Vector Engine Compiler
-g	**-g**
-gv	none
-Ng	**-g0**
-init,stack={ zero | nan | *0xXXXX* **}**	**-minit-stack={ zero | snan | snanf |** *0xXXXX* **}**
-traceback	**-traceback**
-traceback { simple | demangled }	none
-Ntraceback	none

B.1.9 Preprocessor Options

SX Compiler	Vector Engine Compiler
-D*name*[=*def*]	**-D*name*[=*def*]**
-E	**-E**
-EP	none
-H	**-H**
-I *directory-name*	**-I** *directory-name*
-K gcc_predefines	none. Note: Macros are defined by default.
-K nogcc_predefines	none
-Kkeep_comments	**-C**
-Knokeep_comments	none
-Kkeep_line_dirs	none
-Knokeep_line_dirs	none
-Knew_preprocessing	none
-Kold_preprocessing	**-traditional-cpp** Note: -E option is needed.
-Kvariadic_macros	none
-Knovariadic_macros	none

SX Compiler	Vector Engine Compiler
-M	**-M**
-U*name*	**-U*name***
-dD	**-dD**
-dI	**-dI**
-dM	**-dM**
-dN	**-dN**

B.1.10 List Output Options

SX Compiler	Vector Engine Compiler
-Rappend	**-report-append-mode**
-Rnoappend	none
-Rdiaglist	**-report-diagnostics**
-Rnodiaglist	none
-Rfile={ *file-name* | stdout }	**-report-file={ *file-name* | stdout }**
-Rfmtlist	**-report-format**
-Rnofmtlist	none
-Robjlist	**-assembly-list**
-Rnoobjlist	none
-R { summary | nosummary }	none
-Rsystem_header	**-fdiag-system-header**
-R { transform | notransform }	none

B.1.11 Message Options

SX Compiler	Vector Engine Compiler
-O { fullmsg | infomsg | nomsg }	none
-pi { fullmsg | infomsg | nomsg }	**-fdiag-inline={ 2 | 1 | 0 }**
-pvctl { fullmsg | infomsg | nomsg }	**-fdiag-parallel={ 2 | 1 | 0 }** **-fdiag-vector={ 2 | 1 | 0 }**

SX Compiler	Vector Engine Compiler
-wall	**-Wall**
-wno_unset_use	none
-wnone	**-w**
-wfatal=*n*	none
-woff=*n*	none
-wlongjmp	none

B.1.12 Assembler Options

SX Compiler	Vector Engine Compiler
-Wa,*option-strings*	**-Wa,***option-strings*

B.1.13 Linker Options

SX Compiler	Vector Engine Compiler
-G	none
-L*directory-name*	**-L***directory-name*
-l*library-name*	**-l***library-name*
-Wl,*option-strings*	**-Wl,***option-strings*

B.1.14 Directory Options

SX Compiler	Vector Engine Compiler
-YI,*directory-name*	none
-YL,*directory-name*	none
-YS,*directory-name*	none
-Ya,*directory-name*	none
-Yc,*directory-name*	none
-Yl,*directory-name*	none
-Ys,*directory-name*	none

SX Compiler	Vector Engine Compiler
-Yt,*directory-name*	none

B.2 Compiler Directives

Please refer to “C.3 Compiler Directives” to confirm the correspondence tables of compiler directives between SX compilers and compilers for the Vector Engine. Please use the “compiler directive conversion tool” for converting from the SX compiler directive to the Vector Engine. Please refer to “Appendix C Compiler Directive Conversion Tool” for detail.

B.3 Environment Variables

SX Compiler	Vector Engine Compiler
C_PROGINF	**VE_PROGINF**
C_TRACEBACK	**VE_TRACEBACK**

B.4 Implementation-Defined Specifications

B.4.1 Data Types

Type	SX Compiler		Vector Engine Compiler	
	Size	Alignment	Size	Alignment
_Bool	4	4	1	1
bool	4	4	1	1
char **signed char** **unsigned char**	1	1	1	1
short **short int** **unsigned short** **unsigned short int**	2	2	2	2

Type	SX Compiler		Vector Engine Compiler	
	Size	Alignment	Size	Alignment
int **unsigned int**	4	4	4	4
long **long int** **unsigned long** **unsigned long int**	8	8	8	8
long long **long long int** **unsigned long long** **unsigned long long int**	8	8	8	8
float	4	4	4	4
double	8	8	8	8
long double	16	16	16	16
float _Complex	8	4	8	4
double _Complex	16	8	16	8
long double _Complex	32	16	32	16
pointer	8	8	8	8
enum	4	4	4	4
Array type	(*1)	(*2)	(*1)	(*3)
Structure type **union type** **Class type**	(*1)	(*1)	(*1)	(*1)
Bit-fields	4 (*4)	4 (*4)	(*5)	(*5)

(*1) The specifications of SX Compiler and Vector Engine Compiler are the same. See “9.2.1 Data Types”.

(*2) Requires the same size and alignment as the array element, except for the char type array. The char type array requires a 16-byte alignment.

(*3) Requires the same size and alignment as the array element.

(*4) Correspond with **int**.

(*5) Bit-filelds obey the same size and alignment rules as other structure and union members.

B.4.2 Predefined Macros

The following predefined macros of the SX compiler are not defined by the Vector Engine compiler.

Name
__BUILTIN_ABS
_C99
_C99_COMPLEX
_C99LIB
_EXCEPTION_ENABLE
_FLOAT0
_LONG64
_RESTRICT
_SIGNED_CHAR
_SIZE_T64
__STDC_NO_THREADS__
SX
_SX
__SXCXX_EXTENSIONS
__SXCXX_REVISION
_VECLIB

Appendix C Compiler Directive Conversion Tool

This appendix describes the tool for converting from the SX compiler directive to the Vector Engine.

C.1 ncdirconv

Name:

ncdirconv

SYNOPSIS:

ncdirconv [OPTION...] [FILE | DIRECTORY]...

DESCRIPTION:

This tool converts the sxf90/sxf03/sxcc/sxc++ directive to the nfort/ncc/nc++ directive in source file.

When this tool specifies a directory, it convert files with the following extensions in that directory at once.

.c .i .h .C .cc .cpp .cp .cxx .c++ .ii .H .hh .hpp
.hp .hxx .h++ .tcc .F .FOR .FTN .FPP .F90 .F95 .F03 .f
.for .ftn .fpp .f90 .f95 .f03 .i90

The original file is saved as file-name.bak.

The sxf90/sxf03/sxcc/sxc++ directives can be left after conversion or deleted by option.

OPTIONS:

Option	Description
-a, --append	Append the nfort/ncc/nc++ directive. Do not delete the sxf90/sxf03/sxcc/sxc++ directives.
-d, --delete	If the nfort/ncc/nc++ directive is not supported, delete the sxf90/sxf03/sxcc/sxc++ directive.
-f, --force	Do not check file suffix.
-h, --help	Display this help and exit.
-o file, --output file	Specify output file-name. When multiple input files are specified, or when a directory is specified, this option is ignored.
-p, --preserve	If the nfort/ncc/nc++ directive is not supported, do not delete the sxf90/sxf03/sxcc/sxc++ directive.
-q, --quiet	Do not report about conversion.

Option	Description
-r, --recursive	Recursively conversion any subdirectories found.
-v, --version	Output version information and exit.

Messages:

If the compiler directive is converted or the nfort/ncc/nc++ does not support the compiler directive, the message is output to the standard error.

Format:

```
file-name: line Line-number: message
```

file-name: Input file name

Line-number: Line number of file before conversion

message:

- converted "*SX compiler directive*" to "*VE compiler directive*" (Converted | Substitute)

  Indicates that the compiler directive has been converted. "Converted" is output if compiler directive of the SX and VE have equivalent functions. "Substitute" is output if compiler directive of SX and VE have nearly equivalent functions.

- "*SX compiler directive*" is not supported [(Remained| Removed/Obsolescent)]

  The sxf90/sxf03/sxcc/sxc++ directive is not supported by VE. "Remained" is output to the compiler directive scheduled for future implementation in the VE. "Removed/Obsolescent" is output to the compiler directive that is not planned to be supported.

Exit status:

The exit status is 0 if conversion is successful, otherwise it is nonzero.

Notes:

This tool is creates a temporary file for work in /tmp. This temporary file is automatically deleted at the end of the execution. The directory can be changed with the environment variable **TMPDIR**.

C.2 Examples

Example1: When a file specified.

Convert the sxf90/sxf03/sxcc/sxc++ directive contained in a file to the nfort/ncc/nc++ directive.

```
$ cat sample.c
int func(int max)
{
    int i;
    int sum = 0;

#pragma cdir novector
    for (i=0; i<max; i++) {
        sum += i;
    }
    return sum;
}
```

```
$ ncdirconv sample.c
sample.c: line   6: converted 'novector' to 'novector' (Converted)
```

```
$ cat sample.c
int func(int max)
{
    int i;
    int sum = 0;

#pragma _NEC novector
    for (i=0; i<max; i++) {
        sum += i;
    }
    return sum;
}
```

Example2: When a directory is specified.

Take the following directory as an example.

dir/

+ Makefile
+ sample1.c
+ sample2.c
+ subdir/
    + Makefile
    + sample3.c

```
$ ncdirconv dir
dir/sample1.c: line   5: converted 'loopcnt=5' to 'loop_count(5)' (Converted)
dir/sample2.c: line  16: converted 'nodep' to 'ivdep' (Substitute)
```

In the above case, sample1.c and sample2.c are converted. Makefile is out of scope because there is no file extension. Files in subdirectory 'subdir' are also excluded.

```
$ ncdirconv -r dir
dir/sample2.c: line   5: converted 'nodep' to 'ivdep' (Substitute)
dir/sample1.c: line  16: converted 'loopcnt=5' to 'loop_count(5)' (Converted)
dir/subdir/sample3.c: line  12: converted 'loopcnt=5' to 'loop_count(5)'
(Converted)
```

Specify **-r** option to convert files in subdirectories. If **-r** option is specified, directory is recursively checked and converted.

C.3 Compiler Directives

SX Compiler	VE Compiler
alias	(Removed/Obsolescent)
alloc_on_vreg(*identifier*, *n*)	**vreg**(*identifier*)
altcode	**dependency_test** **loop_count_test** **shortloop_reduction**
altcode=dep	**dependency_test**
altcode=loopcnt	**loop_count_test**
altcode=nodep	**nodependency_test**
altcode=noshort	**noshortloop_reduction**
altcode=short	**shortloop_reduction**
noaltcode	**nodependency_test** **noloop_count_test** **noshort_loop_reduction**
assoc	**assoc**
noassoc	**noassoc**
assume	**assume**
noassume	**noassume**
atomic	**atomic**

SX Compiler	VE Compiler
cncall	**cncall**
collapse	**collapse**
compress	(Removed/Obsolescent)
nocompress	(Removed/Obsolescent)
concur	**concurrent**
concur(by=*m*)	**concurrent schedule**(dynamic, *m*)
concur(for=*n*)	**concurrent**
noconcur	**noconcurrent**
data_prefetch	(Removed/Obsolescent)
delinearize	(Removed/Obsolescent)
nodelinearize	(Removed/Obsolescent)
divloop	**vwork**
nodivloop	**novwork**
expand	**unroll_complete**
expand=n	(Removed/Obsolescent)
noexpand	**nounroll**
extend	(Removed/Obsolescent)
extend_free	(Removed/Obsolescent)
fixed	(Removed/Obsolescent)
free	(Removed/Obsolescent)
gthreorder	**gather_reorder**
nogthreorder	(Removed/Obsolescent)
iexpand(*function*)	**inline**
noiexpand(*function*)	**noinline**
inline(*function*)	**inline**
inline(*function***) complete**	**inline_complete**
noinline(*function*)	**noinline**
inner	**inner**
noinner	**noinner**
listvec	**list_vector**

SX Compiler	VE Compiler
nolistvec	**nolist_vector**
loop_eq	(Removed/Obsolescent)
noloop_eq	(Removed/Obsolescent)
loopchg	**interchange**
noloopchg	**nointerchange**
loopcnt=*n*	**loop_count**(*n)*
lstval	**lstval**
nolstval	**nolstval**
move	**move_unsafe**
nomove	**nomove**
nomovediv	**move**
neighbors	(Removed/Obsolescent)
noneighbors	(Removed/Obsolescent)
nexpand	**inline_complete**
noconflict(*identifier*)	(Removed/Obsolescent)
nodep	**ivdep**
on_adb(*identifier*)	(Removed/Obsolescent)
outerunroll=*n*	**outerloop_unroll**(*n*)
noouterunroll	**noouterloop_unroll**
overlap	(Removed/Obsolescent)
nooverlap	(Removed/Obsolescent)
parallel for	**parallel for**
parallel for private(*identifier*)	**parallel for private**(*identifier*)
parallel sections	(Removed/Obsolescent)
section	(Removed/Obsolescent)
select(*keyword*)	(Remained)
shape	(Removed/Obsolescent)
shortloop	**shortloop**
skip	(Removed/Obsolescent)
sparse	(Remained)

SX Compiler	VE Compiler
nosparse	(Remained)
split	(Remained)
nosplit	(Remained)
sync	(Remained)
nosync	**nosync**
threshold	(Removed/Obsolescent)
nothreshold	(Removed/Obsolescent)
traceback	(Remained)
unroll=*n*	**unroll**(*n*)
nounroll	**nounroll**
unshared	(Removed/Obsolescent)
vecthreshold	**vector_threshold**(*n*)
vector	**vector**
novector	**novector**
verrchk	**verror_check**
noverrchk	**noverror_check**
vlchk	(Removed/Obsolescent)
novlchk	(Removed/Obsolescent)
vob	**vob**
novob	**novob**
vovertake(*identifier*)	**vovertake**
novovertake	**novovertake**
vprefetch	(Remained)
novprefetch	(Removed/Obsolescent)
vreg(*identifier*)	**vreg**(*identifier*)
vwork=*keyword*	(Removed/Obsolescent)
vworksz=*n*	(Removed/Obsolescent)

C.4 Notes

- If **-a** or **-p** is specified, the SX compiler directive will remain and a warning will be

output at compile time.

```
$ ncc -c sample.c
"sample.c", line 6: warning: unrecognized #pragma
  #pragma cdir novector
          ^

ncc: vec( 103): sample.c, line 8: Unvectorized loop.
```

- The original file is saved as file-name.bak. When file-name.bak already exists, rename file-name.bak to file-name.bak2, then save the new file as file-name.bak. Up to five files are saved. Please delete files as necessary.
- This tool does not check the format of the input file. If the format of the SX compiler directive is incorrect, conversion may not be performed correctly.
- If the input file is a symbolic link file, the symbolic link destination file is updated. The "file-name.bak" is created as a regular file.

Appendix D Change Notes

The following changes are done from the previous version (Rev.27 Jun.2022 released).

- Added a description of diagnostic message opt(1108) to Chapter 11.

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z