NEC

SX-Aurora TSUBASA

Fortran Compiler User's Guide

SX-Aurora TSUBASA

Proprietary Notice

Remarks:

- This document is the revision 32nd issued in Jul 2023.
- NEC Fortran Compiler conforms to the following language standards.
  - ISO/IEC 1539-1:2010 Programming languages - Fortran
  - OpenMP Application Program Interface Version 4.5
- NEC Fortran compiler also conforms a part of "ISO/IEC 1539-1:2018 Programming languages – Fortran"
- NEC Fortran compiler also conforms a part of "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.

Contents

Chapter1 Fortran Compiler........1
1.1 Overview........1
1.2 Usage of the Compiler........1
1.3 Execution........3
1.4 Command Line Syntax........4
1.5 Specifying Compiler Options........4
1.6 Searching Module Files........5
1.7 Searching files included by INCLUDE line or #include directive........6
1.8 Searching Libraries........6
1.9 Arithmetic Exceptions........6
1.9.1 Operation Result After Arithmetic Exception Occurrence........6
1.9.2 Changing Arithmetic Exception Mask........8
1.9.3 Using Traceback Information........8
1.9.4 Remarks on Changing Arithmetic Exception Mask........9
1.10 Execution Time Termination Codes........9
Chapter2 Environment Variables........10
2.1 Environment Variables Referenced During Compilation........10
2.2 Environment Variables Referenced During Execution........12
Chapter3 Compiler Options........31
3.1 Overall Options........32
3.2 Optimization Options........33
3.3 Parallelization Options........41
3.4 Inlining Options........42
3.5 Code Generation Options........44
3.6 Debugging Options........45
3.7 Language Options........47
3.8 Message Options........49
3.9 List Output Options........50
3.10 Preprocessor Options........51
3.11 Assembler Options........52
3.12 Linker Options........52

3.13 Directory Options 54
3.14 Miscellaneous Options 54
3.15 Compiler options which cannot specify by options directive 55
3.16 Optimization Level and Options' Defaults 56
Chapter4 Compiler Directives 58
Chapter5 Optimization and Vectorization 66
5.1 Code Optimization 66
5.1.1 Optimizations 66
5.1.2 Side Effects of Optimization 67
5.2 Vectorization Features 67
5.2.1 Vectorization 67
5.2.2 Partial Vectorization 68
5.2.3 Optimizing Mask Operations 68
5.2.4 Macro Operations 69
5.2.5 Conditional Vectorization 73
5.2.6 Outer Loop Strip-mining 73
5.2.7 Short-loop 74
5.2.8 Packed vector instructions 75
5.2.9 Other 75
5.2.10 Remarks on Using Vectorization 75
5.3 Other features for performance 77
5.3.1 Offloading of Lumped Output of Array 77
5.3.2 Improve efficiency in buffering 77
Chapter6 Inlining 79
6.1 Automatic Inlining 79
6.2 Explicit Inlining 79
6.2.1 Description 79
6.2.2 Specifying Inline Directive 80
6.2.3 Remarks 80
6.3 Cross-file Inlining 81
6.4 Inline Expansion Inhibitors 82
6.5 Notes on Inlining 82
6.6 Restrictions on Inlining 83
Chapter7 Parallelization 84

7.1 Automatic Parallelization 84
7.1.1 Description 84
7.1.2 Conditional Parallelization Using Threshold Test 84
7.1.3 Conditional Parallelization Using Dependency Test 84
7.1.4 Parallelization of inner Loops 84
7.1.5 Forced Loop Parallelization 85
7.2 OpenMP Parallelization 86
7.2.1 Using OpenMP Parallelization 86
7.2.2 OpenMP 5.0 86
7.2.3 Extensions on OpenMP Parallelization 86
7.2.4 Restrictions on OpenMP Parallelization 87
7.2.5 Using OpenMP Parallelization 88
7.3 Threads 88
7.3.1 Set and Get Number of Threads 88
7.3.2 Thread Creation and Destroy 89
7.3.3 Postpone Thread Creation 90
7.4 Notes on Using Parallelization 90
Chapter8 Compiler Listing 91
8.1 Option List 91
8.2 Diagnostic List 91
8.2.1 Format of Diagnostic List 91
8.2.2 Notes 92
8.3 Format List 92
8.3.1 Format of Format List 93
8.3.2 Loop Structure and Vectorization/Parallelization/Inlining Statuses ... 93
8.3.3 Notes 96
8.4 Optimization List of Each Module 96
8.4.1 Inlining Module 96
8.4.2 Vectorization Module 97
8.4.3 Code Generation Module 97
Chapter9 Programming Notes Depending on the Language Specification 100
9.1 Non-Standard Extended Features 100
9.1.1 Statements 100
9.1.2 Program 109

9.1.3 Source Form 110
9.1.4 Expressions.. 110
9.1.5 Deleted Features 112
9.2 Implementation-Defined Specifications 113
9.2.1 Data Types 113
9.2.2 Internal Representation of Data 113
9.2.3 Specifications 122
9.2.4 Predefined Macro 122
9.2.5 Notes for Intrinsic Procedures 124
9.3 Memory Allocation and Deallocation 124
9.3.1 Memory block 124
9.3.2 Change size and threshold size of memory block 125
9.4 Run-Time Input/Output 125
9.4.1 Formatted Records 125
9.4.2 Unformatted Records 126
9.4.3 Preconnection 129
9.4.4 Unnamed File 130
9.4.5 Rounding Mode 130
9.4.6 NAMELIST Input Format 131
9.4.7 NAMELIST Output Format 131
9.5 Fortran 2018 Extensions 132
9.5.1 Execution Control 132
9.5.2 Intrinsic Procedures and Modules 132
9.5.3 Input/Output 134
9.5.4 Programs and Procedures 134
9.5.5 Language-Mixed Programming 135
9.5.6 Obsolescent features 137
9.6 Restrictions 137
Chapter10 Language-Mixed Programming 138
10.1 Point of Mixed Language Programming 138
10.2 Correspondence of C/C++ Function Name and Fortran Procedure Name
139
10.2.1 External Symbol Name of Fortran Procedure 139
10.2.2 External Symbol Name of C++ Function 140

10.2.3 Rules for Corresponding C/C++ Functions with Fortran Procedures 141
10.2.4 Examples of Calling 141
10.3 Data Types 144
10.3.1 Integer and Logical Types for Fortran 144
10.3.2 Floating-point and Complex Types for Fortran 145
10.3.3 Character Type for Fortran 146
10.3.4 Derived Type for Fortran 146
10.3.5 Pointer 147
10.3.6 Common Block for Fortran 149
10.3.7 Notes 150
10.4 Type and Return Value of Function and Procedure 150
10.5 Passing Arguments 153
10.5.1 Fortran Procedure Arguments 153
10.5.2 Notes 155
10.6 Linking 157
10.6.1 Linking Fortran Program and C Program 157
10.6.2 Linking Fortran Program and C++ Program 157
10.7 Notes 157
Chapter11 Library Reference 158
11.1 Intrinsic Procedures 158
11.1.1 ABS(*A*) Specific Name 158
11.1.2 ACOS(*X*) Specific Name 159
11.1.3 ACOSH(*X*) Specific Name 159
11.1.4 AIMAG(*Z*) Specific Name 160
11.1.5 AINT(*A*) Specific Name 160
11.1.6 AMT(*X*) 161
11.1.7 AND(*I*,*J*) 161
11.1.8 ANINT(*A*) Specific Name 162
11.1.9 ASIN(*X*) Specific Name 162
11.1.10 ASINH(*X*) Specific Name 163
11.1.11 ATAN(*X*) Specific Name 163
11.1.12 ATAN2(*Y*,*X*) Specific Name 164
11.1.13 ATANH(*X*) Specific Name 164

11.1.14 BTEST(*I*,*POS*) Specific Name 165
11.1.15 CANG(*X*) 166
11.1.16 CBRT(*X*) 166
11.1.17 CLOCK(*D*) 167
11.1.18 CONJG(*Z*) Specific Name 167
11.1.19 COS(*X*) Specific Name 167
11.1.20 COSD(*X*) 168
11.1.21 COSH(*X*) Specific Name 169
11.1.22 COTAN(*X*) 169
11.1.23 DATE(*A*) 170
11.1.24 DATIM(*A*,*B*,*C*) 170
11.1.25 DBLE(*A*) Specific Name 171
11.1.26 DCMPLX(*X*,*Y*) 171
11.1.27 DFACT(*I*) 172
11.1.28 DFLOAT(*A*) 172
11.1.29 DIM(*X*,*Y*) Specific Name 173
11.1.30 DREAL(*A*) 173
11.1.31 ERF(*X*) Specific Name 174
11.1.32 ERFC(*X*) Specific Name 174
11.1.33 ETIME(*D*) 175
11.1.34 EXIT(*X*) 175
11.1.35 EXP(*X*) Specific Name 176
11.1.36 EXP10(*X*) 176
11.1.37 EXP2(*X*) 177
11.1.38 EXPC(*X*) 177
11.1.39 EXPC10(*X*) 178
11.1.40 EXPC2(*X*) 178
11.1.41 FACT(*I*) 179
11.1.42 FLUSH(*UNIT*) 179
11.1.43 GAMMA(*X*) Specific Name 180
11.1.44 IAND(*I*,*J*) Specific Name 180
11.1.45 IBCLR(*I*,*POS*) Specific Name 181
11.1.46 IBITS(*I*,*POS*,*LEN*) Specific Name 182
11.1.47 IBSET(*I*,*POS*) Specific Name 182

11.1.48 IEOR(*I*,*J*) Specific Name 183
11.1.49 IMAG(*A*) 184
11.1.50 INT(*A*[,*KIND*]) Specific Name 184
11.1.51 IOR(*I*,*J*) Specific Name 185
11.1.52 IRE(*X*) 186
11.1.53 ISHFT(*I*,*SHIFT*) Specific Name 187
11.1.54 ISHFT(*I*,*SHIFT*[,*SIZE*]) Specific Name 187
11.1.55 ISNAN(*X*) 188
11.1.56 IXOR(*I*,*J*) 188
11.1.57 LGAMMA(*X*) 188
11.1.58 LOC(*X*) 189
11.1.59 LOG(*X*) Specific Name 189
11.1.60 LOG10(*X*) Specific Name 190
11.1.61 LOG2(*X*) 191
11.1.62 MAX(*A1*,*A2*[,*A3*,⋯]) Specific Name 191
11.1.63 MAXVL() 192
11.1.64 MIN(*A1*,*A2*[,*A3*,⋯]) 192
11.1.65 MOD(*A*,*P*) Specific Name 193
11.1.66 MVBITS(*FROM*,*FROMPOS*,*LEN*,*TO*,*TOPOS*) Specific Name 194
11.1.67 NINT(*A*[,*KIND*]) Specific Name 195
11.1.68 NOT(*I*) 196
11.1.69 OR(*I*,*J*) 196
11.1.70 QCMPLX(*X*,*Y*) 196
11.1.71 QEXT(*X*) 197
11.1.72 QFACT(*I*) 197
11.1.73 QFLOAT(*A*) 198
11.1.74 QREAL(*A*) 198
11.1.75 REAL(*A*[,*KIND*]) 198
11.1.76 RSQRT(*X*) 199
11.1.77 SIGN(*A*,*B*) Specific Name 200
11.1.78 SIN(*X*) Specific Name 200
11.1.79 SIND(*X*) 201
11.1.80 SINH(*X*) Specific Name 202
11.1.81 SQRT(*X*) Specific Name 202

11.1.82 TAN(*X*) Specific Name 203
11.1.83 TANH(*X*) Specific Name 204
11.1.84 TIME(*A*) 204
11.1.85 XOR(*I*,*J*) 205
11.2 Matrix Multiply Library 205
11.2.1 MATRIX-VECTOR Multiplication(*A*, *NAR*, *B*, *NBR*, *C*) 205
11.2.2 MATRIX-VECTOR Multiplication(*A*, *NA*, *IAD*, *B*, *NB*, *C*, *NC*, *NAR*, NBR) 206
11.2.3 MATRIX- MATRIX Multiplication(*A*, *NA*, *IAD*, *B*, *NB*, *IBD*, *C*, *NC*, *ICD*, *NAR*, *NAC*, *NBC*) 208
11.3 UNIX System Function Interface 210
11.3.1 F90_UNIX 212
11.3.2 F90_UNIX_DIR 213
11.3.3 F90_UNIX_ENV 215
11.3.4 F90_UNIX_ERRNO 217
11.3.5 F90_UNIX_FILE 218
11.3.6 F90_UNIX_PROC 222
11.4 Other Library 225
11.4.1 ABORT() 226
11.4.2 ACCESS(*PATH,MODE*) 226
11.4.3 ALARM(*SECS,PROC*) 226
11.4.4 CHDIR(*PATH*) 227
11.4.5 CHMOD(*NAME,MODE*) 227
11.4.6 CTIME(*I*) 228
11.4.7 DTIME(*TARRAY*) 228
11.4.8 ETIME(*TARRAY*) 228
11.4.9 FDATE() 229
11.4.10 FORK() 229
11.4.11 FREE(*ADDR*) 230
11.4.12 FREE2(*ADDR*) 230
11.4.13 FSEEK(*UNIT,OFFSET,WHENCE*) 230
11.4.14 FSTAT(*UNIT,SXBUF*) 231
11.4.15 FTELL(*UNIT*) 232
11.4.16 FTELLI8(*UNIT*) 232

11.4.17 GETARG(*POS,VAL*) 233
11.4.18 GETCWD(*PATH*) 233
11.4.19 GETENV(*NAME,VAL*) 233
11.4.20 GETGID() 234
11.4.21 GETLOG(*NAME*) 234
11.4.22 GETPID() 234
11.4.23 GETPOS(*UNIT*) 235
11.4.24 GETPOSI8(*UNIT*) 235
11.4.25 GETUID() 235
11.4.26 GMTIME(*I,IA9*) 236
11.4.27 HOSTNM(*NAME*) 236
11.4.28 IARGC() 236
11.4.29 IDATE(*IA3*) 237
11.4.30 IERRNO() 237
11.4.31 ISATTY(*UNIT*) 237
11.4.32 ITIME(*IA3*) 238
11.4.33 KILL(*PID,SIGNUM*) 238
11.4.34 LINK(*PATH1,PATH2*) 238
11.4.35 LSTAT(*PATH,SXBUF*) 239
11.4.36 LTIME(*I,IA9*) 240
11.4.37 MALLOC(*SIZE*) 240
11.4.38 MALLOC2(*SIZE*) 241
11.4.39 PERROR(*A*) 241
11.4.40 RENAME(*FROM,TO*) 241
11.4.41 SECNDS(*T*) 242
11.4.42 SIGNAL(*SIGNUM,HANDLER*) 242
11.4.43 SLEEP(*SECS*) 243
11.4.44 STAT(*UNIT,SXBUF*) 243
11.4.45 SYMLNK(*PATH1,PATH2*) 244
11.4.46 SYSTEM(*CMD*) 244
11.4.47 TIME() 245
11.4.48 TTYNAM(*UNIT*) 245
11.4.49 UNLINK(*PATH*) 245
11.4.50 WAIT(*STATUS*) 246

11.5 Notes 246
Chapter12 Messages 248
12.1 Diagnostic Messages 248
12.1.1 Diagnostic Message Format 248
12.1.2 Message List 249
12.2 Runtime Error Messages 260
12.2.1 Format 260
12.2.2 List of Error Messages 260
12.3 Other Runtime Error 289
Chapter13 Troubleshooting 291
13.1 Troubleshooting for compilation 291
13.2 Troubleshooting for execution 297
13.3 Troubleshooting for tuning 301
13.4 Troubleshooting for installation 302
13.5 Troubleshooting for SX-ACE compiler migration 302
Chapter14 VE1/VE3 Compatibility 306
14.1 Executables Compatibility 306
14.2 Changes of Search Path 306
14.3 Changes of Compiler Options 307
14.4 Half-Precision Floating-Point Type 308
14.4.1 Format of Half-Precision Floating-Point Type 308
14.4.2 Mixing binary16 and bfloat16 308
14.5 Notice 308
Chapter15 Notice 310
Appendix A Configuration file 312
A.1 Overview 312
A.2 Format 313
A.3 Example 313
Appendix B SX Compatibility 314
B.1 NEC Fortran 2003 Compiler Options 314
B.1.1 Overall Options 314
B.1.2 Vector/Scalar Optimization Options 315
B.1.3 Inlining Options 318
B.1.4 Parallelization Options 319

B.1.5 Code Generation Options 319
B.1.6 Language Options 320
B.1.7 Performance Measurement Options 321
B.1.8 Debug Options 321
B.1.9 Preprocessor Options 321
B.1.10 List Output Options 322
B.1.11 Message Options 322
B.1.12 Assembler Option 323
B.1.13 C Compiler Option 323
B.1.14 Linker Options 323
B.1.15 Directory Options 324
B.2 FORTRAN90/SX Compiler 324
B.2.1 f90/sxf90 command Options 324
B.2.2 f90/sxf90 Detailed Options for optimization 328
B.2.3 f90/sxf90 Detailed Options for vectorization and parallelization 330
B.2.4 f90/sxf90 Other Detailed Options 333
B.3 Compiler Directives 336
B.4 Environment Variables 336
B.5 Other Library 337
B.6 Implementation-Defined Specifications 339
B.6.1 Data Types 339
B.6.2 Specifications 340
B.6.3 Intrinsic Procedures 340
Appendix C Compiler Directive Conversion Tool 341
C.1 nfdirconv 341
C.2 Examples 342
C.3 Compiler Directives 344
C.4 Notes 347
Appendix D File I/O Analysis Information 348
D.1 Output Example 348
D.2 Description of items 349
Appendix E Change Notes 354
Index 355

Chapter1　Fortran Compiler

1.1 Overview

The NEC Fortran compiler is a compiler that compiles and links Fortran 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 Fortran compiler, set the path to the environment variable **PATH**. The NEC Fortran compiler is installed by default under /opt/nec/ve. Add /opt/nec/ve/bin to the environment variable **PATH**.

Although the NEC Fortran compiler provides environment variables for setting paths such as header files and libraries, the NEC Fortran compiler automatically searches for the default path, so you can use it without setting these environment variables. 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 “Chapter2 Environment Variables”.

(2) Examples

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

- Compiling and linking a Fortran source file (a.f90).

```
$ nfort a.f90
```

- Compiling and linking more than one source file.

```
$ nfort a.f90 b.f90
```

- Compiling, linking, and naming an executable file.

```
$ nfort -o prog.out a.f90
```

- Compiling and linking with the highest vectorization and optimization.

```
$ nfort -O4 a.f90
```

- Compiling and linking with safe vectorization and optimization.

```
$ nfort -O1 a.f90
```

- Compiling and linking without vectorization and optimization.

```
$ nfort -O0 a.f90
```

- Compiling and linking using automatic parallelization.

```
$ nfort -mparallel a.f90
```

- Compiling and linking using automatic inlining.

```
$ nfort -finline-functions a.f90
```

- Compiling and linking using the half-precision floating-point. (VE3 only)

– IEEE binary16 format

```
$ nfort a.f90
```

– bfloat16 format

```
$ nfort -mfp16-format=bfloat a.f90
```

- Compiling and linking using a compiler of specific version.

```
$ /opt/nec/ve/bin/nfort-X.X.X a.f90          (X.X.X is version number.)
```

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 data.in 10           (input the file ”data.in” and value ”10”)
```

- Executing with redirecting an input file.

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

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

```
$ nfort -mparallel -O3 a.f90 b.f90
$ export OMP_NUM_THREADS=4
$ ./a.out
```

- Executing with connecting a file to unit.

```
$ export VE_FORT9=DATA9       (connect the file ”DATA9” to unit number 9)
$ ./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.

```
$ nfort -pg a.f90
$ ./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.

```
nfort [ 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:

```
$ nfort -v -c a.f90          (Correct)
$ nfort -vc a.f90            (Incorrect)
```

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

Suffix	Recognized File
.F .FOR .FTN .FPP .fpp	Fortran source file (Fixed form, With preprocessing)
.F90 .F95 .F03	Fortran source file (Free form, With preprocessing)
.f .for .ftn .i	Fortran source file (Fixed form, Without preprocessing)
.f90 .f95 .f03 .i90	Fortran source file (Free form, Without preprocessing)
.c	C source 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 Fortran Compiler. Compiler options and files can be specified in the same way as when the command line is used. The option file must be placed in the home directory, to which the environment variable **HOME** has been set.

Compiler Type	Option File Name
nfort	$HOME/.nfortinit

Example:

```
$ cat ~/.nfortinit
-O3 -finline-functions
$ nfort -v a.f90
/opt/nec/ve/libexec/fcom … -O3 -finline-functions … a.f90
```

1.6 Searching Module Files

When there are modules in an input source file, in order that other source files refer to the modules, the Fortran compiler outputs compiled module information files for each modules. The compiled module information files of the intrinsic modules are beforehand prepared in the defined place.

(1) Searching compiled module information files of non-intrinsic module

When there are not modules which are referred to in an input source file, the Fortran compiler searches the following directories in the following order for module files:

(a) Directory on which each input source file is

(b) Directories specified by **-module**

(c) Current directory

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

(e) Subdirectory named “include” under the directory specified by **-B**

(f) Directories specified by the environment variable **NFORT_INCLUDE_PATH**

(g) Directory specified by **-isystem**

(h) /opt/nec/ve/nfort/<*version-number*>/include

(i) /opt/nec/ve/include (When **-march**=ve3 is enabled: /opt/nec/ve3/include)
When **-isysroot** is enabled, subdirectory named “include” under the directory specified by **-isysroot**.

(2) Searching compiled module information files of intrinsic modules

The intrinsic modules are referred to by **USE** statement with **INTRINSIC** attribute. The Fortran compiler searches the following directory for intrinsic module files:

(a) Directory specified by **-fintrinsic-modules-path** if it is specified, otherwise /opt/nec/ve/nfort/<*version-number*>/include

1.7 Searching files included by INCLUDE line or #include directive

The Fortran compiler searches the following directories in the following order for files included by **INCLUDE** line and **#include**"file-name".

(a) Directory on which each input source file is
(b) Current directory
(c) Directories specified by **-I**
(d) Subdirectory named “include” under the directory specified by **-B**
(e) Directories specified by the environment variable **NFORT_INCLUDE_PATH**
(f) Directory specified by **-isystem**
(g) /opt/nec/ve/nfort/<*version-number*>/include
(h) /opt/nec/ve/include (When **-march**=ve3 is enabled: /opt/nec/ve3/include)
When **-isysroot** is enabled, subdirectory named “include” under the directory specified by **-isysroot**.

1.8 Searching Libraries

The Fortran 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 **NFORT_LIBRARY_PATH**
(d) /opt/nec/ve/nfort/<*version-number*>/lib
(When **-march**=ve3 is enabled: /opt/nec/ve3/nfort/<*version-number*>/lib)
(e) Directories specified by the environment variable **VE_LIBRARY_PATH**
(f) /opt/nec/ve/lib/gcc (When **-march**=ve3 is enabled: /opt/nec/ve3/lib/gcc)
(g) /opt/nec/ve/lib (When **-march**=ve3 is enabled: /opt/nec/ve3/lib)

1.9 Arithmetic Exceptions

1.9.1 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.

1.9.2 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.

1.9.3 Using Traceback Information

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

Example:

```
$ nfort -traceback=verbose below.f90 out.f90 watch.f90 hey.f90 ovf.f90
...
$ export VE_TRACEBACK=VERBOSE
$ export VE_FPE_ENABLE=DIV
$ ./a.out
Runtime Error: Divide by zero at 0x600008001088
[ 0] 0x600008001088 below_          below.f90:3
[ 1] 0x600018001168 out_            out.f90:3
[ 2] 0x600020001168 watch_          watch.f90:3
[ 3] 0x600010001168 hey_            hey.f90:3
[ 4] 0x60000001cab8 MAIN__          ovf.f90:5
```

In example, the exception of “Divide by zero” occurred in line 3 of below.f90.

1.9.4 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.

1.10 Execution Time Termination Codes

Termination Codes when the program ends are listed below.

Termination Code	Meaning
0	Normal termination.
1	Execution-time error.
2	If character-type termination code is specified in the **ERROR STOP** statement, it is used as the termination code.
137	Execution-time error (Abort).
N	If a termination code *n* is specified in the **STOP** statement or the intrinsic subroutine **EXIT**, it is used as the termination code.

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.

NFORT_COMPILER_PATH

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

Example:

```
$ export NFORT_COMPILER_PATH= “$HOME/libexec:$HOME/wk/libexec”
```

NFORT_INCLUDE_PATH

Specifies a list of directories separated by colon which are searched for the files included by **INCLUDE** line or **#include** directive, and module 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.

Example:

```
$ export NFORT_INCLUDE_PATH= “$HOME/include:$HOME/wk/include”
```

NFORT_LIBRARY_PATH

Specifies a list of directories separated by colon which are searched for the Fortran 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 Fortran compiler.

Example:

```
$ export NFORT_LIBRARY_PATH= “$HOME/lib”
```

NFORT_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 Fortran 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 NFORT_PROGRAM_PATH= "$HOME/bin:$HOME/wk/bin"
```

PATH

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

LD_LIBRARY_PATH

Specifies a directory where the Library for offloading of lumped and formatted output of array, and lumped and list-directed output of array to VH is put.

Example:

```
$ export LD_LIBRARY_PATH=/opt/nec/ve/nfort/lib64
```

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 specify the suffixes “B”(Bytes), “K”(Kilobytes), “M”(Megabytes), and “G”(Gigabytes) as unit. The stack size used by each threads is 4 megabytes when it is not specified explicitly.

Example:

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

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_ERRCTL_ALLOCATE

This variable is used to control the program execution when a runtime error related to allocation of an allocatable variable or a pointer occurs.

One of the following values can be specified.

ABORT

The program is aborted with error message. (default)

MSG

Error message is output and the execution is continued if possible.

NOMSG

No error message is output and the execution is continued if possible.

Example:

```
$ export VE_ERRCTL_ALLOCATE=MSG
```

VE_ERRCTL_DEALLOCATE

This variable is used to control the program execution when a runtime error related to deallocation of an allocatable variable or a pointer occurs.

One of the following values can be specified.

ABORT

The program is aborted with error message.

MSG

Error message is output and the execution is continued if possible.

NOMSG

No error message is output and the execution is continued if possible. (default)

Example:

```
$ export VE_ERRCTL_DEALLOCATE=ABORT
```

VE_FMTIO_OFFLOAD

This variable controls offloading of lumped and formatted output of array, and lumped and list-directed output of array. When the value of this variable is “**YES”** or “**ON”**, offloading is enabled. See the

VE_FMTIO_OFFLOAD_THRESHOLD

This variable sets the threshold of the number of array element offloading of lumped and formatted output of array, and lumped and list-directed output of array. An array which have element smaller than the specified value is not offloaded to VH. The default value is 10.

Example:

```
$ export VE_FMTIO_OFFLOAD_THRESHOLD=20
```

VE_FORT*n*

This variable sets a file name to be connected to the unit number *n*.
Default of the file name is fort.*n*.
If this variable is set, a file name is changed to its value.

Example:

```
$ export VE_FORT9=DATA9
```

VE_FORT_ABORT

This variable controls core dump creation if a fatal error occurs. When the value of this variable is “YES”, core dump is created.

Note　This variable does not control core dump creation other than caused by "Runtime Error" of Fortran.

Example:

```
$ export VE_FORT_ABORT=YES
```

VE_FORT_ACCUMULATE_THREAD_CPU_TIME

This variable is used to control value of **CPU_TIME** subroutine in multithreaded program. When the value of this variable is "YES", then the value is accumulated CPU time of all threads.

Example:

```
$ export VE_FORT_ACCUMULATE_THREAD_CPU_TIME=YES
```

VE_FORT_DEFAULTFILE

This variable is used to control the starting position of default directory path for input/output file from the current directory to the specified directory. When the **FILE** specifier of the **OPEN** statement, or environment variable **VE_FORT***n* are specified by the absolute pathname, the specified environment variable is ignored. If a default directory pathname string does not end in a slash (/), a slash is added. Default value is current directory.

The pathname used for each combination of the specified values is shown below.

FILE specifier in OPEN statement	**VE_FORT_DEFAULTFILE**	**VE_FORT*n***	**Pathname**
none	none	none	./fort.*n*
none	none	test.dat	./test.dat
none	ignored	/usr/tmp/t.dat	/usr/tmp/t.dat
none	/tmp	none	/tmp/fort.*n*
none	/tmp	testdata	/tmp/testdata
none	/usr	lib/testdata	/usr/lib/testdata
file.dat	/usr/group	ignored	/usr/group/file.dat
/tmp/file.dat	ignored	ignored	/tmp/file.dat
file.dat	none	ignored	./file.dat

Example:

```
$ export VE_FORT_DEFAULTFILE=/foo/
```

VE_FORT_EXPRCW

This variable sets the unit number of unformatted file to be treated as a file in the expanded format. Records whose size is over 2GB can be handled in the expanded format. Its format is as follows.

ALL

Apply to all unit numbers.

decimal* | *decimal,decimal* | *decimal-decimal

Apply to the unit decimal.

Apply to the multiple units decimal and decimal.

Apply to the multiple unit from decimal to decimal.

YES[:***decimal***] | NO[:***decimal***]

Specify the expanded format of the file.

Specify units after the colon.

;

Specify exception mode and units.

Example1: Apply to the unit 10.

```
$ export VE_FORT_EXPRCW=10
```

Example2: Apply to the unit 10 and 11.

```
$ export VE_FORT_EXPRCW=10,11
```

Example3: Apply to the unit 10, 11 and 12.

```
$ export VE_FORT_EXPRCW=10-12
```

Example4: Treats all unit as the expanded format except for 10, 11 and 12.

```
$ export VE_FORT_EXPRCW=YES;NO:10-12
```

Example5: Apply to all unit numbers.

```
$ export VE_FORT_EXPRCW=YES
```

VE_FORT_FILEINF

When "YES" or "DETAIL" is set, information about I/O statement execution is output to the standard error output at the file close. The items output here provide information about whether I/O operations are performed as scheduled, and whether there are unit numbers whose performance should be improved, and other information. When display items (such as paths) contain multi-byte characters, it may not be displayed correctly. See Section Appendix D for details.

Example:

```
$ export VE_FORT_FILEINF=DETAIL
```

VE_FORT_FMT_NO_WRAP_MARGIN

This variable is used to control the wrap of list-directed output. When the value of this variable is "YES", column is not wrapped up to maximum record length.

Example:

```
$ export VE_FORT_FMT_NO_WRAP_MARGIN=YES
```

VE_FORT_FMTBUF[*n*]

Sets the size, in bytes, of recode buffers allocated for I/O. **VE_FORT_FMTBUF** can specify the value used for all unit identifiers or one unit identifiers. The buffer size must be 135 or larger. If a value less than 135 is specified, the value is set to 135. When **VE_FORT_FMTBUF** is not set, the buffers size is a value specified in a **RECL** specifier in **OPEN** statement. When **VE_FORT_FMTBUF** and **RECL** specifier is set, the buffers size is a smaller value of either **VE_FORT_FMTBUF** or value of **RECL** specifier. If this variable is specified for the standard input/output file and the standard error output file, this option is ignored.

When **VE_FORT_FMTBUF** and **VE_FORT_RECORDBUF** is set, the priority is as follows.

Highest	**VE_FORT_RECORDBUF***u*	Specifies one unit identifier.
|	**VE_FORT_FMTBUF***u*	Specifies one unit identifier.
|	**VE_FORT_RECORDBUF**	Specifies all unit identifiers.
Lowest	**VE_FORT_FMTBUF**	Specifies all unit identifiers.

The default recode buffers size for I/O is the following value.

- Standard input/output file and Stream file

  65536 Byte

- Sequential file

  65536 Byte or Value of **RECL** specifier

- Direct file

  Value of **RECL** specifier

Example1: for all unit identifiers

```
$ export VE_FORT_FMTBUF=32768
```

Example2: for unit identifier 1

```
$ export VE_FORT_FMTBUF1=60000
```

VE_FORT_FOR_PRINT

This variable sets an output file name for **PRINT** statement or **WRITE** statement with an asterisk (*) in place of a unit number. When it is not specified explicitly, output to standard output.

```
$ export VE_FORT_FOR_PRINT=FILENAME
```

Note When you use this environment variable, an unused logical unit number is automatically assigned. This unit number is represented by a negative number, such as in error messages.

VE_FORT_FOR_READ

This variable sets an input file name for **READ** statement when an asterisk (*) is specified instead of the unit number or the unit number is omitted. When it is not specified explicitly, input from standard input.

```
$ export VE_FORT_FOR_READ=FILENAME
```

Note When you use this environment variable, an unused logical unit number is automatically assigned. This unit number is represented by a negative number, such as in error messages.

VE_FORT_FOR_TYPE

This variable sets an output file name for **TYPE** statement. When it is not specified explicitly, output to standard output.

```
$ export VE_FORT_FOR_TYPE=FILENAME
```

Note When you use this environment variable, an unused logical unit number is automatically assigned. This unit number is represented by a negative number, such as in error messages.

VE_FORT_MEM_BLOCKSIZE

This variable is set the block size of a memory block which is allocated to accelerate memory allocation/deallocation at the beginning of program by the megabytes. The value can be specified as megabytes by using “M” as unit and gigabytes by using “G” as unit. The value must be power of 2. The size is set 64 megabytes when it is not specified explicitly. For each process, three memory blocks is allocated at the beginning of program execution.

Example: Set 16 megabytes

```
$ export VE_FORT_MEM_BLOCKSIZE=16M
```

VE_FORT_NML_DELIM_BLANK

This variable is used to control NAMELIST output of character-type array when **DELIM** specifier is omitted. When "YES" is set, output characters are separated by a blank character. By default ("NO"), output characters are not separated from each other by value separators and are output continuously.

This variable is ignored when **DELIM** specifier is specified.

Example:

```
$ ./a.out
&NML
 C = abcdefg
 /
$ export VE_FORT_NML_DELIM_BLANK=YES
$ ./a.out
```

```
&NML C = a b c d e f g/
```

VE_FORT_NML_REPEAT_FORM

This variable is used to control NAMELIST output of two or more consecutive values in array. By default ("YES"), the same value will be output collectively form (Repeat * Value). When "NO" is set, the values will be output not collectively.

Note The array values are output not collectively, when versions 3.0.7 and earlier.

Example:

```
$ ./a.out
&NML
 R = 3*1.0000000,     2.0000000,     3.0000000
 /
$ export VE_FORT_NML_REPEAT_FORM=NO
$ ./a.out
&NML R =    1.0000000    1.0000000    1.0000000    2.0000000    3.0000000/
```

VE_FORT_NORCW

This variable sets the unit number of unformatted file to be treated as a format to which no control record is added. This option is handled faster than standard record format because recode is treated same as stream file.
The restrictions that apply are that the length of an input record must match the length of the output record or an abnormal result is detected, and the **BACKSPACE** statement cannot be used.

Its format is as follows.

ALL

Apply to all unit numbers.

decimal* | *decimal,decimal* | *decimal-decimal

Apply to the unit decimal.

Apply to the multiple units decimal and decimal.

Apply to the multiple unit from decimal to decimal.

YES[:***decimal***] | NO[:***decimal***]

Specify a format to which no control record is added.

Specify units after the colon.

;

Specify exception mode and units.

Example1: Apply to the unit 10.

```
$ export VE_FORT_NORCW=10
```

Example2: Apply to the unit 10 and 11.

```
$ export VE_FORT_NORCW=10,11
```

Example3: Apply to the unit 10, 11 and 12.

```
$ export VE_FORT_NORCW=10-12
```

Example4: Treats all unit as a format to which no control record is added except for 10, 11 and 12.

```
$ export VE_FORT_NORCW=YES;NO:10-12
```

Example5: Apply to all unit numbers.

```
$ export VE_FORT_NORCW=ALL
```

VE_FORT_PARTRCW

This variable sets the unit number of unformatted file to be treated as a format to which control record is changed. The length of an input record must match the length of the output record or an error is detected.

Its format is as follows.

ALL

Apply to all unit numbers.

decimal* | *decimal,decimal* | *decimal-decimal

Apply to the unit decimal.

Apply to the multiple units decimal and decimal.

Apply to the multiple unit from decimal to decimal.

YES[:***decimal***] | NO[:***decimal***]

Specify a format to which control record is changed.

Specify units after the colon.

;

Specify exception mode and units.

Example1: Apply to the unit 10.

```
$ export VE_FORT_PARTRCW=10
```

Example2: Apply to the unit 10 and 11.

```
$ export VE_FORT_PARTRCW=10,11
```

Example3: Apply to the unit 10, 11 and 12.

```
$ export VE_FORT_PARTRCW=10-12
```

Example4: Treats all unit as a format to which control record is changed except for 10, 11 and 12.

```
$ export VE_FORT_PARTRCW=YES;NO:10-12
```

Example5: Apply to all unit numbers.

```
$ export VE_FORT_PARTRCW=ALL
```

VE_FORT_PAUSE

Determines if a **PAUSE** statement is executed. When a value "NO" is set, ignore a **PAUSE** statement.

Example:

```
$ export VE_FORT_PAUSE=NO
```

VE_FORT_RECLUNIT

This variable sets unit of **RECL** specifier in an **OPEN** statement for unformatted file. For units, you can specify only **“**BYTE” or **“**WORD”. Default unit is **“**BYTE”. **“**WORD” is 4-byte cycle.

Example:

```
$ export VE_FORT_RECLUNIT=WORD
```

VE_FORT_RECORDBUF[*n*]

Sets the size, in bytes, of recode buffers allocated for I/O.

VE_FORT_RECORDBUF can specify the value used for all unit identifiers or one unit identifiers. The buffer size must be 135 or larger. If a value less than 135 is specified, the value is set to 135. When **VE_FORT_RECORDBUF** is not set, the buffers size is a value specified in a **RECL** specifier in **OPEN** statement. When **VE_FORT_RECORDBUF** and **RECL** specifier is set, the buffers size is a smaller value of either **VE_FORT_RECORDBUF** or value of **RECL** specifier. If this variable is specified for the standard input/output file and the standard error output file, this option is ignored.

When **VE_FORT_FMTBUF** and **VE_FORT_RECORDBUF** is set, the priority is as follows.

Highest	**VE_FORT_RECORDBUF***u*	Specifies one unit identifier.
|	**VE_FORT_FMTBUF***u*	Specifies one unit identifier.
|	**VE_FORT_RECORDBUF**	Specifies all unit identifiers.
Lowest	**VE_FORT_FMTBUF**	Specifies all unit identifiers.

The default recode buffers size for I/O is the following value.

- Standard input/output file and Stream file

  65536 Byte

- Sequential file

  65536 Byte or Value of **RECL** specifier

- Direct file

  Value of **RECL** specifier

Example1: for all unit identifiers

```
$ export VE_FORT_RECORDBUF=32768
```

Example2: for unit identifier 1

```
$ export VE_FORT_RECORDBUF1=60000
```

VE_FORT_SETBUF[*n*]

Sets the size, in kilobytes, of an I/O buffers allocated for I/O. **VE_FORT_SETBUF** can specify the value used for all unit identifiers or one unit identifiers. If this variable is specified for the standard input/output file and the standard error output file, this option is ignored except for specifying 0 to the standard output and standard error output file. When **VE_FORT_SETBUF** is not set, the size of an I/O buffers is the following value.

- Sequential file and Stream file
  - Record buffer environment variable value is less than or equal to 512KB

    512 KB
  - Record buffer environment variable value is greater than 512KB

    Raise fractions of Record buffer environment variable value to unit (KB)
- Direct file
  - Record length is less than or equal to 4,096 bytes

    4 KB
  - Record length is greater than 2,048,000,000 bytes

    2,000,000 KB
  - Other record length

    Raise fractions of record length to unit (KB)

Note The above "Record buffer environment variable value" is the value set to **VE_FORT_FMTBUF** or **VE_FORT_RECORDBUF**.

Example1: for all unit identifiers

```
$ export VE_FORT_SETBUF=10
```

Example2: for unit identifier 1

```
$ export VE_FORT_SETBUF1=20
```

VE_FORT_SUBRCW

This variable sets the unit number of unformatted file to be treated as a file in the format divided into records. Records whose size is over 2GB can be handled in the expanded format.

When any of **VE_FORT_EXPRCW**, **VE_FORT_NORCW** or **VE_FORT_PARTRCW** is set, this variable is ignored.

Its format is as follows.

ALL

Apply to all unit numbers.

decimal* | *decimal*,*decimal* | *decimal*-*decimal

Apply to the unit decimal.

Apply to the multiple units decimal and decimal.

Apply to the multiple unit from decimal to decimal.

YES[:***decimal***] | NO[:***decimal***]

Specify a file in the format divided into records.

Specify units after the colon.

;

Specify exception mode and units.

Example1: Apply to the unit 10.

```
$ export VE_FORT_SUBRCW=10
```

Example2: Apply to the unit 10 and 11.

```
$ export VE_FORT_SUBRCW=10,11
```

Example3: Apply to the unit 10, 11 and 12.

```
$ export VE_FORT_SUBRCW=10-12
```

Example4: Treats all unit as a file in the format divided into records except for 10, 11 and 12.

```
$ export VE_FORT_SUBRCW=YES;NO:10-12
```

Example5: Apply to all unit numbers.

```
$ export VE_FORT_SUBRCW=ALL
```

VE_FORT_UFMTADJUST[*n*]

This variable is used to control adjust the length of list item at input/output. **VE_FORT_UFMTADJUST** can specify the value used for all unit identifiers or one unit identifiers. When this variable is set, then the different kind of data than the kind of input/output list item type can input/output.

The following values can be specified. Two or more values can be specified by comma delimitation.

ALL

Same as VE_FORT_UFMTADJUST=INT,LOG,REAL,DBL.

DBL

If the kind of input/output list item type is REAL(16) or COMPLEX(16), the kind on the file regard as REAL(8) or COMPLEX(8).

INT

If the kind of input/output list item type is INTEGER(8), the kind on the file regard as INTEGER(4).

LOG

If the kind of input/output list item type is LOGICAL(8), the kind on the file regard as LOGICAL(4).

NO

No adjust the length.

REAL

If the kind of input/output list item type is REAL(8) or COMPLEX(8), the kind on the file regard as REAL(4) or COMPLEX(4).

Example1: Apply adjust the length of all type to the unit 10.

```
$ export VE_FORT_UFMTADJUST10=ALL
```

Example2: Apply adjust the length of all type to all unit except the unit 10.

```
$ export VE_FORT_UFMTADJUST=ALL
$ export VE_FORT_UFMTADJUST10=NO
```

Example3: Apply adjust the length of real and complex to the unit 10.

```
$ export VE_FORT_UFMTADJUST10=REAL,DBL
```

VE_FORT_UFMTENDIAN

This variable sets the unit number of unformatted file to be treated as a file in the big-endian format. Its format is as follows.

ALL

Apply to all unit numbers.

decimal* | *decimal,decimal* | *decimal-decimal

Apply to the unit decimal.

Apply to the multiple units decimal and decimal.

Apply to the multiple unit from decimal to decimal.

big[:***decimal***] | little[:***decimal***]

Specify the endian format of the file.

Specify units after the colon.

;

Specify exception mode and units.

Example1: Apply to the unit 10.

```
$ export VE_FORT_UFMTENDIAN=10
```

Example2: Apply to the unit 10 and 11.

```
$ export VE_FORT_UFMTENDIAN=10,11
```

Example3: Apply to the unit 10, 11 and 12.

```
$ export VE_FORT_UFMTENDIAN=10-12
```

Example4: Treats all unit as big endian except for 10, 11 and 12.

```
$ export VE_FORT_UFMTENDIAN=big;little:10-12
```

Example5: Apply to all unit numbers.

```
$ export VE_FORT_UFMTENDIAN=ALL
```

VE_FORT_UFMTENDIAN_NOVEC

This variable sets the unit number of unformatted file to be treated as a file in the big-endian format and the conversion should be done by the scalar operation. Its format is as follows.

ALL

Apply to all unit numbers.

decimal* | *decimal*,*decimal* | *decimal*-*decimal

Apply to the unit decimal.

Apply to the multiple units decimal and decimal.

Apply to the multiple unit from decimal to decimal.

YES[:***decimal***] | NO[:***decimal***]

Specify a file in the big-endian format to be converted by the scalar operation.

Specify units after the colon.

;

Specify exception mode and units.

Example1: Apply to the unit 10.

```
$ export VE_FORT_UFMTENDIAN_NOVEC=10
```

Example2: Apply to the unit 10 and 11.

```
$ export VE_FORT_UFMTENDIAN_NOVEC=10,11
```

Example3: Apply to the unit 10, 11 and 12.

```
$ export VE_FORT_UFMTENDIAN_NOVEC=10-12
```

Example4: Treats all unit except for 10, 11 and 12 as a file in the big-endian format to be converted by the scalar operation.

```
$ export VE_FORT_UFMTENDIAN_NOVEC=YES;NO:10-12
```

Example5: Apply to all unit numbers.

```
$ export VE_FORT_UFMTENDIAN_NOVEC=ALL
```

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 following values can be specified. Two or more values can be specified by comma delimitation.

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_HEAP

This variable sets the value to initialize the heap area at the run-time. When the value is not set, the heap area is not initialized.

The following values can be specified.

ZERO

Initializes with zeros.

NAN

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

NANF

Initializes with quiet NaN in single precision (0x7fffffff).

SNAN

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

SNANF

Initializes with signaling NaN in single precision (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:

```
$ export VE_INIT_HEAP=ZERO
```

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 precision (0x7fffffff7fffffff).

NANF

Initializes with quiet NaN in single precision (0x7fffffff).

SNAN

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

SNANF

Initializes with signaling NaN in single precision (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:

```
$ nfort -minit-stack=runtime a.f90
$ 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 Fortran 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:

```
$ nfort -traceback a.f90
...
$ export VE_TRACEBACK=FULL
$ export VE_FPE_ENABLE=DIV
Runtime Error: Divide by zero at 0x600000000cc0
[ 1] Called from 0x7f5ca0062f60
[ 2] Called from 0x600000000b70
Floating point exception
```

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
Runtime Error: Overflow at 0x600008001088
[ 0] 0x600008001088 below_          below.f90:3
[ 1] 0x600018001168 out_            out.f90:3
[ 2] 0x600020001168 watch_          watch.f90:3
[ 3] 0x600010001168 hey_            hey.f90:3
[ 4] 0x60000001cab8 MAIN__          ovf.f90:5
```

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 Fortran program using the Fortran compiler system.

The compiler options of the Fortran compiler can be divided into the following categories.

- Overall Options
  Compiler options used to control the Fortran 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*.

f77

Compiles as a Fortran source file of fixed form.

f77-cpp-input

Does preprocessing and compiles as a Fortran source file of fixed form.

f95

Compiles as a Fortran source file of free form.

f95-cpp-input

Does preprocessing and compiles as a Fortran source file of free form.

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.

-fargument-noalias

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

-f[no-]associative-math

Allows [Disallows] re-association of operands in series during optimization and loop transformation. When **-fno-associative-math** is specified, the optimization which transforms 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 [Disallows] aggressive re-association of operands in series during optimization and loop transformation. (default: **-fno-aggressive-associative-math**)

-f[no-]assume-contiguous

Allows [Disallows] the compiler to assume that assumed-shape array is contiguous.

(default:**-fno-assume-contiguous**)

-f[no-]copyin-intent-out

[Dose not] Create copy-in operation for an argument which has **INTENT(OUT)** attribute. (default: **-fcopyin-intent-out**)

-f[no-]cse-after-vectorization

[Does not] Re-apply common subexpression elimination after vectorization.

(default: **-fno-cse-after-vectorization**)

-f[no-]fast-formatted-io

[Does not] Use fast version formatted I/O.

(default: **-ffast-formatted-io**)

-f[no-]fast-math

[Does not] Uses fast scalar version math functions outside of vectorized loops.

(default: **-ffast-math**)

-f[no-]fast-math-check

[Does not] Checks the value ranges of arguments in the mathematical function's fast scalar version.

(default: **-fno-fast-math-check**)

-f[no-]ignore-asynchronous

[Does not] Ignores **ASYNCHRONOUS** attribute in optimization.

(default: **-fno-ignore-asynchronous**)

-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 an external-routine 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-complete-nest=*m*

Unrolls loops except for the outermost loop by *m* level nesting when complete loop unrolling is applied.

Unrolls from 1 to *m*-dimension of an array expression when complete loop unrolling is applied. (default: **-floop-unroll-complete-nest**=3)

Remark:

-floop-unroll-completely-nest=*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

Enables [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

Allows [Disallows] 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**).

-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)

-fnamed-noalias-aggressive

The compiler will assume that the object pointed-to-by a named pointer does not have an alias in applying optimization and vectorization. This option applies optimization and vectorization aggressively.

-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 version of power operation when the exponent is an integer value. The result becomes more exact but the calculation speed becomes 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-]reorder-logical-expression

Allows [Disallows] evaluate the terms in a logical expression from left to right order instead of any order. (default: **-freorder-logical-expression**)

-f[no-]replace-loop-equation

[Does not] Replaces “!=”, “==”, “.NE.” and “.EQ.” operator with “<=” or “>=” at the loop back-edge. (default: **-fno-replace-loop-equation**)

-f[no-]replace-matmul-to-matrix-multiply

Allows [Disallows] to replace MATMUL call into a vector matrix library function call. (default: **-freplace-matmul-to-matrix-multiply**)

-m[no-]array-io

Allows [Disallows] to optimize array expression and “implied DO” in I/O statement. (default: **-marray-io**)

-m[no-]conditional-index-test

Allows [Disallows] to conditional-index-testing optimization. (default: **-mno-conditional-index-test**)

-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.

-mstack-arrays

Allocates automatic arrays and temporary arrays on the stack. (default)

-mno-stack-arrays

Allocates automatic arrays and temporary arrays on in heap memory.

-muse-mmap

Use mmap / munmap functions to allocate / deallocate memory in **ALLOCATE** / **DEALLOCATE** statements.

-m[no-]vector

Enables [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)

-mvector-assignment-threshold=*n*

Use vector instructions to assign a derived type whose size is equal to or greater than *n* byte. (default: **-mvector-assignment-threshold**=64)

-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 and intrinsic arithmetic in the vectorized version. (default: **-mno-vector-intrinsic-check**)

The target mathematical functions and intrinsic arithmetic of this option are as follows. The argument is restricted to double precision real type and specific name which have the type is also target.

ACOS, ACOSH, ASIN, ATAN, ATAN2, ATANH, COS, COSD, COSH, COTAN, EXP, EXP10, EXP2, EXPC, FACT, LOG10, LOG2, LOG, SIN, SIND, SINH, TAN, TANH, Exponentiation

-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-neighbors

Allows [Disallows] neighboring access optimization.

(default: **-mno-vector-neighbors**)

-mvector-neighbors is available when **-march**=ve3 is enabled.

-m[no-]vector-packed

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

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

Allows [Disallows] to replace R1**R2 in a vectorized loop with EXP(R2*LOG(R1)). R1 and R2 type must be single or double precision floating-point type. 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 R1**R2 in a vectorized loop with the expression including SQRT or CBRT when R2 is a special value such as 0.5, 1.0/3.0 etc. R1 and R2 type must be single or double precision floating-point type. When it is replaced, the execution time would become faster, but numerical error occurs rarely in the calculation.

(default: **-mvector-power-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-vecvtor-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.

-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 executionat 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

-f[no-]inline-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-copy-arguments

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

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

-finline-directory=*directory*

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

-fno-inline-directory=*directory*

Does not search all source files under directories separated by colon for routines 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 routines 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 routines 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 routines 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 routine size (= the amount of intermediate representations for a routine) to be inlined by automatic inlining.
(default: **-finline-max-function-size**=50)

-finline-max-times=*n*

Sets the limit of the route size (= the amount of intermediate representations for a routine) after automatic inlining to “(routine-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*.fil".

-mread-il-file *IL file name*

Read IL files separated by colon for routines 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**)

-march=*kind*

Specifies the target architecture.

The following are available as *kind*:

ve1

Produces object files available only on ve1 or later. (default)

ve3

Produces object files available only on ve3 or later.

(Defaults when installed for VE3.)

-mfp16-format=*kind*

Specifies format of the half-precision floating-point. **-mfp16-format**=*kind* can be specified only when **-march**=ve3 is enabled.

The following are available as *kind*:

none

Does not use format of the half-precision floating-point.

ieee

Uses IEEE binary16 format.

bfloat

Uses bfloat16 format.

-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

-fbounds-check

Same as **-fcheck=bounds**.

-fcheck=*keyword*

Enables runtime check according to *keyword*. Two or more keywords can be specified by separating them with a colon (:). For example, if you specify this option as "-fcheck=all:noalias", all checks except alias can be enabled.

The following are available as *keyword*:

all

Enables checking all keywords below.

[no]alias

Enables [Disables] checking assignments to aliased dummy arguments.

[no]bits

Enables [Disables] checking bit intrinsic arguments.

[no]bounds

Enables [Disables] checking array bounds.

[no]dangling

Enables [Disables] checking for dangling pointers.

[no]do

Enables [Disables] checking DO loops for zero step values.

[no]iovf

Enables [Disables] checking integer overflow.

[no]pointer

Enables [Disables] checking pointer references.

[no]present

Enables [Disables] checking optional references.

[no]recursion

Enables [Disables] checking for invalid recursion.

-g

Generates debugging information in DWARF. When **-O1**, **-O2**, **-O3**, or **-O4** are specified with **-g**, some of the debugging information may be inaccurate as a side-effect of optimization.

-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 precision (0x7fffffff7fffffff).

nanf

Initializes with quiet NaN in single precision (0x7fffffff).

snan

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

snanf

Initializes with signaling NaN in single precision (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.

-mmemory-trace

Generates code to output memory allocation/deallocation trace.

-mmemory-trace-full

Generates code to output memory allocation/deallocation trace with source code information.

-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

-bss

Allocates local variables and arrays in .bss section.

-fdefault-integer=*n*

Specifies the size of default **INTEGER** and **LOGICAL** in byte. *n* must be 4 or 8. (default: **-fdefault-integer**=4)

It also affects the intrinsic procedures that the result type or argument type is default **INTEGER** or default **LOGICAL**. The result or argument type must be of one of the following types:

- default **INTEGER**
  - *n*=4: default **INTEGER** or **INTEGER**(4)
  - *n*=8: default **INTEGER** or **INTEGER**(8)
- default **LOGICAL**
  - *n*=4: default **LOGICAL** or **LOGICAL**(4)
  - *n*=8: default **LOGICAL** or **LOGICAL**(8)

-fdefault-double=*n*

Specifies the size of default **DOUBLE PRECISION** and real/imaginary parts of **DOUBLE COMPLEX** in byte. *n* must be 8 or 16. (default: **-fdefault-double**=8)

-fdefault-real=*n*

Specifies the size of default **REAL** and real/imaginary parts of default **COMPLEX** in byte. *n* must be 4 or 8. (default: **-fdefault-real**=4)

It also affects the intrinsic procedures that the result type or argument type is default **REAL** or default **COMPLEX**. The result or argument type must be of one of the following types:

- default **REAL**
  - *n*=4: default **REAL** or **REAL**(4)
  - *n*=8: default **REAL** or **REAL**(8)
- default **COMPLEX**
  - *n*=4: default **COMPLEX** or **COMPLEX**(4)
  - *n*=8: default **COMPLEX** or **COMPLEX**(8)

-fextend-source

Extends the limit of 72 characters on a source line in fixed form to 2,048.

-ffree-form

Specifies that the input source program is described in free form. This is the default when the suffix of input source file is **.f90**, **.f95**, **.f03**, **.F90**, **.F95** or **.F03**.

-ffixed-form

Specifies that the input source program is described in fixed form. This is the default when the suffix of input source file is **.f** or **.F**.

-ff90-sign

Does not distinguish the second argument of the intrinsic function SIGN between positive real 0.0 and negative real -0.0. If the second argument is negative real -0.0, sign of the result value is positive.

-fmax-continuation-lines=*n*

Specifies the upper limit of the number of lines is designated. *n* must be 511 or upper and 4095 or lower. (default: **-fmax-continuation-lines**=1023)

-fno-realloc-lhs

Enables **-fno-realloc-lhs-array** and **-fno-realloc-lhs-scalar** at the same time. (default: **-frealloc-lhs**)

-fno-realloc-lhs-array

By Fortran 2003 standard, when the left-hand side of an assignment is an allocatable array variable and it is unallocated or not allocated with the correct shape to hold the right-hand side, it should be reallocated to the shape of the right-hand side.

This option specifies ignoring the rule. When the left-hand side is not allocated with the correct shape to hold the right-hand side, it causes unexpected result. (default: **-frealloc-lhs-array**)

-fno-realloc-lhs-scalar

By Fortran 2003 standard, when the left-hand side of an assignment is an allocatable scalar variable and it is unallocated, it should be automatically reallocated.

This option specifies ignoring the rule. When the left-hand side is not allocated, it causes unexpected result.

(default: **-frealloc-lhs-scalar**)

-masync-io

Specifies that the data transfer occur asynchronously when ASYNCHRONOUS=”YES” in the READ and WRITE statement is specified.

Asynchronous I/O is enabled with the following I/O.

- Unformatted I/O.

-save

Treats each program unit (except those marked as **RECURSIVE**) as if **SAVE** statement were specified for every local variable.

-std=*standard*

Specifies Fortran Language standard. The recognized keywords are f95, f2003, f2008 or f2018. (default:**-std**=f2008)

-use *module*

References all public entities within module accessible. Two or more module can be specified by comma delimitation.

3.8 Message Options

-Wall

Outputs all syntax warning messages.

-Werror

Treats all syntax warnings as fatal errors.

-Wextension

Outputs a warning message for use of extended Fortran language specification.

-Wobsolescent

Outputs a warning message for use of obsolescent Fortran language specification.

-Woverflow

Outputs a warning message for integer overflow at the compilation.

-Woverflow-errors

Outputs an error message for integer overflow and stop the compilation.

-Wunmatched-subscript

Outputs a warning message at the compilation, when the number of subscript expression in subscript list of the array reference is smaller than the rank of array.

-Wunmatched-subscript-errors

Output an error message and stop the compilation, when the number of subscript expression in subscript list of the array reference is smaller than the rank of array.

-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)

-pedantic-errors

Outputs the errors for deviation from language specification.

-w

Suppresses all syntax 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-vector

[Does not] Outputs optimization list of vectorization module.

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

3.10 Preprocessor Options

-D*macro*[=*defn*]

Defines *macro* as the value *defn* as if **#define** directive 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.

-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.

-fpp

Specifies that the input source program is preprocessed by **fpp** before the compilation. This is the default when the suffix of input source file is **.F**, **.F90**, **.F95** or **.F03**.

-nofpp

Specifies that the input source program is not preprocessed by **fpp** before the compilation. This is the default when the suffix of input source file is **.f**, **.f90**, **.f95** or **.f03**.

-fpp-name=*name*

Specifies the *name* (which can be either with or without a pathname) of Fortran preprocessor to be used instead of the default one.

-I*directory*

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

-isysroot *directory*

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

-isystem *directory*

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

-M

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

-nostdinc

Omits searching the standard system directory for header files.

-P

Omits outputting line directives to preprocessed text.

-traditional

Specifies to remove C-style comment (/**/) completely instead of replacing with a space.

-U*macro*

Undefines the definition of macro.

-Wp,*option*

Specifies *option* to be passed to preprocessor (**fpp**). 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

-cxxlib

Link the C++ libraries.

-Bdynamic

Enables the linking of dynamic-link libraries at the run-time. (default)

-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.

-shared

Generates a shared object.

-static

Link libraries statically.

-static-nec

Link the NEC SDK libraries statically.

-stdlib=*library-name*

Specifies the linked C++ library when **-cxxlib** is specified. The following libraries can be specified.

compat

Link NEC Compat C++ Standard Library.

(default when NEC Compat C++ Standard Library is installed.)

libc++

Link libc++.

(default when NEC Compat C++ Standard Library is NOT installed.)

-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.

-fintrinsic-modules-path *directory*

Specifies a *directory* name where intrinsic module files are searched for.

-module *directory*

-J *directory*

Specifies a *directory* name where to output module files. The specified *directory* is also added to the list of searching path which is used during inputting module 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***

- Optimization Options

  -muse-mmap

- Parallelization Options

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

- Inlining Options

  -finline-abort-at-error, **-mgenerate-il-file** *IL file name*

- Code Generation Options

  -no-proginf

- Debugging Options

  -mmemory-trace, **-mmemory-trace-full**, **-traceback**

- Language Options

  -masync-io, **-use** *module*

- Message Options

  -Werror

- Preprocessor Options

  -D*macro*[=*defn*], **-E**, **-fpp**, **-nofpp**, **-fpp-name**=*name*, **-M**, **-P**, **-U***macro*,

-traditional, **-Wp**,*option*

- Assembler Options

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

- Linker Options

  -Bdynamic, **-Bstatic**, **-L***directory*, **-l***library*, **-nostartfiles**, **-nostdlib**, **-rdynamic**, **-shared**, **-static**, **-static-nec**, **-stdlib**, **-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
-fassociative-math	✓	✓	✓	-	-
-ffast-math	✓	✓	✓	✓	-
-fignore-induction-variable-overflow	✓	-	-	-	-
-fignore-volatile	✓	-	-	-	-
-finline-copy-arguments	-	✓	✓	✓	✓
-floop-collapse	✓	✓	-	-	-
-floop-fusion	✓	✓	-	-	-
-floop-interchange	✓	✓	-	-	-
-floop-normalize	✓	✓	-	-	-

Option Name	-O4	-O3	-O2	-O1	-O0
-floop-strip-mine	✓	✓	-	-	-
-floop-unroll	✓	✓	✓	-	-
-floop-unroll-complete=4	✓	✓	✓	-	-
-floop-unroll-complete-nest=3	✓	✓	✓	-	-
-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	✓	-	-	-	-
-freplace-matmul-to-matrix-multiply	✓	✓	✓	✓	-
-marray-io	✓	✓	✓	✓	-
-mconditional-index-test	✓	✓	-	-	-
-msched-none	-	-	-	-	✓
-msched-block	✓	✓	✓	✓	-
-mvector	✓	✓	✓	✓	-
-mvector-dependency-test	✓	✓	✓	-	-
-mvector-fma	✓	✓	✓	-	-
-mvector-merge-conditional	✓	✓	-	-	-

Chapter4 Compiler Directives

This chapter describes the compiler directives of Fortran compiler. Its format is as follows.

Format:

!NEC$ *directive-name* [*clause*] ... (Free source form) ***NEC$** *directive-name* [*clause*] ... (Fixed source form) **cNEC$** *directive-name* [*clause*] ... (Fixed source form)

Note The following formats are also available, but marked obsolescent. The above formats are recommended.

!$NEC *directive-name* [*clause*] ... (Free source form) ***$NEC** *directive-name* [*clause*] ... (Fixed source form) **c$NEC** *directive-name* [*clause*] ... (Fixed source form)

[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 routine which includes this directive should be always inlined. This directive must be specified in a called routine. A routine call which **noinline** is effective is never inlined even if the called routine 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 assignment statement immediately after the compiler directive to which **atomic** is specified is reduction operation such as summation or product.

cncall

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

collapse

Allows loop collapsing.

[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)

dependency_test

Allows 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 routine 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 routine includes a routine call, the called routine is chosen as a candidate for inlining. The inlining applied until there is no routine 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.

loop_count_test

Allows the conditional vectorization by loop-iteration-count-test.

[no]lstval

Allows [Disallows] 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.

[no]neighbors

Allows [Disallows] neighboring access optimization in the loop.

Neighboring access optimization is effective only when **-march=**ve3 is enabled.

nofma

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

nofuse

Disallows the loop fusion with the previos 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 and **#line** 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 options directive cannot be specified in file included by **INCLUDE** line.
- 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 [Disallows] to use packed vector instruction in the loop.

parallel do

Applies forced-parallelization of the following loop. The programmer must check the validity of the operation when the loop is parallelized. **-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)

The **private**-clause whose functionality is the same as OpenMP can be specified. You can specify a scalar variable and/or explicit-shaped array whose type is not **CHARACTER** or derived type.

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 **INTEGER(KIND=4)**, **REAL(KIND=4)**, or their alias names.
- 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. **-fassociative-math** must be effective.

[no]sparse

sparse

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

nosparse

Assumes that the number of mathematical intrinsic 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.

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 an array expression or **DO** loop.

[no]vob

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

[no]vovertake

Allows [Disallows] all vector stores in the array expression or 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 **DO** 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 **INTEGER(KIND=4)**, **INTEGER(KIND=8)**, **REAL(KIND=4)**, **REAL(KIND=8)**, or their alias names.
- 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 Fortran 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))
- Optimization of arithmetic IF statements (**-O**[*n*] (*n*=1,2,3,4))
- 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))
- Optimization of implied DO lists in an I/O statement (**-O**[*n*] (*n*=1,2,3,4), **-marray-io**)
- 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 Optimizing Mask Operations

Using masked operations makes vectorization possible for a DO loop containing an **IF** statement. However, if **IF** statements are nested to make a complex condition, identical operations may arise between masks, lowering execution efficiency. In order to avoid this, optimization is performed as follows for mask operations when **-O**[*n*] (*n*=1,2,3,4) is valid.

- Process identical operations as common expressions

  In this example, "A(I).LE.0.0" is processed as a common expression.

  Example:

```
DO I = 1, N
    IF (A(I).LE.0.0)THEN
        X(I) = A(I) * B(I)
    END IF
    Y(I) = A(I) + B(I)
    IF (A(I).LE.0.0.AND.B(I).EQ.0.0) THEN
        Z(I) = A(I)
    END IF
END DO
```

  (Vectorization)

```
M1i    = 0: if Ai > 0.0
         1: if Ai <= 0.0
```

```
Xi     = Ai * Bi (if M1i = 1)
Yi= Ai * Bi
M2i    = 0: if Bi ≠ 0.0
1: if Bi = 0.0
M3i    = M1i AND M2i
Zi= Ai (if M3i = 1)
```

- When **IF** statements are nested to make a complex condition, perform common expression processing. This vectorization is performed when **-O**[*n*] (*n*=1,2,3,4) is valid.

  In this example, "Y(I).GT.0.0" is processed as a common expression.

  Example:

```
DO I = 1, N
    IF (X(I).GT.0.0) THEN
        IF (Y(I).GT.0.0) THEN
            Z(I) = Y(I) / X(I)
        ELSE
            Z(I) = 0.0
        END IF
    ELSE
        IF (Y(I).GT.0.0) THEN
            Z(I) = X(I) / Y(I)
        END IF
    END IF
END DO
```

(Vectorization)

```
M1i    = 0: if Xi <= 0.0
         1: if Xi >  0.0
M2i    = 0: if Yi <= 0.0
         1: if Yi >  0.0
M3i    = M1i AND M2i
Zi     = Yi / Xi (if M3i = 1)
M4i    = M1i AND M2i
Zi     = 0.0 (if M4i = 1)
M5i    = M1i AND M2i
Zi= Yi / Xi (if M5i = 1)
```

5.2.4　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
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
t2 = t1 * exp2
    ...
S = tn * expn
```

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

- Iteration

```
A(I) = exp ± A(I-1)           (exp: An expression)
A(I) = exp * 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)
A(I) = t * exp2
```

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

- Maximum values and minimum values
  - Function type

Example:

```
DO I = 1, N
```

```
    XMAX = MAX(XMAX, X(I))
END DO
```

– Finding the maximum or minimum value only

Example:

```
DO I = 1, N
    IF (XMAX .LT. X(I)) THEN
        XMAX = X(I)
    END IF
END DO
```

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

Example:

```
DO I = 1, N
    IF (XMIN .GT. X(I)) THEN
        XMIN = X(I)
        IX = I
    END IF
END DO
```

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

Example:

```
DO J = 1, N
    DO I = 1, N
        IF (XMIN .GT. X(I, J)) THEN
            XMIN = X(I, J)
            IX = I
            IY = J
        END IF
    END DO
END DO
```

– Compares absolute values

Example:

```
DO I = 1, N
    IF (ABS(XMIN) .GT. ABS(X(I))) THEN
        XMIN = X(I)
    END IF
END DO
```

- Search

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

  Example:

```
DO I = 1, N
    IF (X(I) .EQ. 0.0) THEN
        EXIT
    END IF
END DO
```

  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
DO I = 1, N
    IF (X(I) .GT. 0.0) THEN
        J = J + 1
        Y(J) = Z(I)
    END IF
END DO
```

- Expansion

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

  Example:

```
J = 0
DO I = 1, N
    IF (X(I) .GT. 0.0) THEN
        J = J + 1
        Z(I) = Y(J)
```

```
    END IF
END DO
```

5.2.5 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) is valid.

The compiler option and the compiler directive option which controls this vectorization is following.

Condition	Compiler Option	Compiler Directive Option
Data dependency	-mvector-dependency-test	dependency_test
Loop iteration	-mvector-loop-count-test	loop_count_test
Loop iteration for reduction operation	-mvector-shortloop-reduction	[no]shortloop_reduction

5.2.6 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 A "tightly nested loop" is a nested loop, in which there is no executable statement between each of **DO** statements nor between each of **ENDDO** statements as shown in Example below.

Example: Tightly nested loop

```
DO I = 1, 10
    DO J = 1, 1000
        A(J) = A(J) + B(J, I) * C(J, I)
    ENDDO
ENDDO
```

Example: Not tightly nested loop (Statement exists between each of **DO** statements)

```
DO K=1,10
    D(K)=0.0
    DO J=1,20
        DO I=1,30
            A(I,J,K)=B(I,J,K)*C(I,J,K)
        ENDDO
        X(K,J)=Y(K,J)+Z(K,J)
    ENDDO
ENDDO
```

Example: Not tightly nested loop (Other loop exists between each of **ENDDO** statements)

```
DO K=1,10
    DO J=1,20
        DO I=1,10
            S(I,J,K)=T(I,J,K)*U(I,J,K)
        ENDDO
        DO I=1,30
            A(I,J,K)=B(I,J,K)*C(I,J,K)
        ENDDO
    ENDDO
ENDDO
```

5.2.7　Short-loop

A loop code which omits the determination of loop termination 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**.

5.2.8 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**.

5.2.9 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))
- Recognition matrix multiply loop (**-O**[*n*] (*n*=3,4), **-fassociative-math**, **-fmatrix-multiply**)
- Loop expansion (**-O**[*n*] (*n*=2,3,4), **-floop-unroll-complete**=*m*)

5.2.10 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 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 the loop containing reference and definition of the array element is vectorized, exception can occur. In such a case, specify **-mno-vector** to stop vectorization or **!NEC$ NOVECTOR** before the loop. The data cannot satisfy vectorizable alignment is dummy argument. The compiler supposes the

dummy data satisfy vectorizable argument and vectorize it.

5.3 Other features for performance

5.3.1 Offloading of Lumped Output of Array

Lumped and formatted output of arrays, and lumped and list-directed output of arrays are offloaded to VH to improving the performance of execution. Set the environment variable **VE_FMTIO_OFFLOAD** to **YES** or **ON**, and set the environment variable **LD_LIBRARY_PATH** to /opt/nec/ve/nfort/lib64 to use this feature.

Example: Lumped and Formatted Output of Array

```
SUBROUTINE FUN
INTEGER I(100)
I=100
WRITE(*,'(I5)') I
```

END **Example:** Lumped and List-Directed Output of Array

```
SUBROUTINE FUN
INTEGER I(100)
I=100
WRITE(*,*) I
END
```

5.3.2 Improve efficiency in buffering

Unformatted I/O in a sequential file access may be improving the performance of I/O by changing record and I/O buffer size.

5.3.2.1 Record buffer

Unformatted I/O in a sequential file access uses the record buffer for I/O-list and data transfer. Therefore, I/O performance can improve by allocating the record buffer larger than the maximum record. Use the environment variable **VE_FORT_RECORDBUF** to change the record buffer size.

5.3.2.2 I/O buffer

File I/O transfers data between the file and the I/O buffer. The file system has an optimal data transfer size. Therefore, I/O performance can improve by allocating the I/O buffer size to the optimal data transfer size. Also, I/O performance can improve by allocating the I/O buffer size larger than the file size when the memory size is acceptable. Use the environment variable **VE_FORT_SETBUF** to change the I/O buffer size.

Chapter6　Inlining

6.1 Automatic Inlining

When automatic inlining is enabled, the compiler chooses the appropriate procedures by analyzing the source files and inline 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 routine 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 routine which includes this directive should be always inlined. This directive must be specified in a called routine. A routine call has **noinline** is never inlined even if the called routine includes this directive.
- **inline**
  A routine 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 routine includes a routine call, the called routine is chosen as a candidate for inlining. The inlining applied until there is no routine calls if possible.
- **noinline**
  A routine call in a following statement, a compound statement, an iteration statement, or a selection statement is never inlined. The routine which includes **always_inline** is not inlined, too.

6.2.2 Specifying Inline Directive

(1) Called routine

always_inline must be specified in a called routine.

```
      SUBROUTINE SUB
!NEC$ ALWAYS_INLINE
      ...
      END SUBROUTINE
```

(2) Statement

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

```
!NEC$ INLINE
    X = FUNC1(A) + FUNC2(A)
    Y = FUNC3(A)
```

FUNC1() and FUNC2() are candidates for inlining, but FUNC3() is not.

(3) **BLOCK** construct, **DO** construct, and **IF** construct

inline / **inline_complete** / **noinline** affect all routine calls in a following construct.

```
!NEC$ INLINE
    DO I=1,N
        CALL SUB1
        CALL SUB2
    END DO
```

Subroutine SUB1 and SUB2 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.
- The routine definition which includes **always_inline** is not removed.
- A routine call which **noinline** is effective is not inlined even if the called routine includes **always_inline**.
- A **BLOCK** construct, **DO** construct, or **IF** construct includes a construct and each construct has opposite directive, the immediately before directive is effective for

the inner construct.

```
!NEC$ INLINE
    BLOCK
        CALL SUB1           ! Candidate for inlining
!NEC$ NOINLINE
        BLOCK
            CALL SUB2       ! Not inlined
        END BLOCK
    END BLOCK
```

6.3 Cross-file Inlining

The compiler inlines procedures 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 procedures to inline are specified.

The following examples show how to specify the source files.

- A source file is specified.

```
$ nfort -c -finline-functions -finline-file=sub.f90 call.f90
```

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

```
$ nfort -c -finline-functions -finline-file=sub2.f90:all call.f90 sub.f90
```

- All source files under a directory are specified.

```
$ ls dir
sub.f90 sub2.f90 sub3.f90
$ nfort -c -finline-functions -finline-directory=dir sub.f90
```

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

```
$ ls dir
sub.f90 sub2.f90 sub3.f90
$ nfort -c -finline-functions -finline-directory=dir -fno-inline-file=sub2.f90
call.f90
```

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.

```
$ nfort -mgenerate-il-file sub.f90
```

```
$ nfort -c -finline-functions -mread-il-file sub.fil main.f90
```

6.4 Inline Expansion Inhibitors

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

- The procedure to be inlined cannot be located.
- The arguments used in the calling sequence do not match the arguments in the procedure to be inlined.
- There is a conflict between common blocks of the calling procedure and the procedure to be inlined.
- The procedure to be inlined contains a NAMELIST input/output statement.
- The procedure to be inlined contains variables having **SAVE** attribute.
- A function name referenced in the procedure to be inlined conflicts with a non-function name used in the calling procedure.
- The procedure to be inlined contains OpenMP directives.
- The procedure to be inlined contains a recursive call of it.

6.5 Notes on Inlining

- If inlining is applied to too many procedures 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 procedures to be inlined carefully.
- A procedure 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.

6.6 Restrictions on Inlining

- In cross-file inlining, the compiler does not search a source file when it contains an **EQUIVALENCE** statement where a thread private common block appears
- In cross-file inlining, module procedures which refer to variables with **PRIVATE** attributes cannot be inlined.

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 nonparallelized 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

!NEC$ PARALLEL DO parallelizes a **DO**-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 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. *variable* must be a scalar variable or an explicit-shaped array whose type is not **CHARACTER** or derived type.

PRIVATE(variable[,variable]...)

!NEC$ ATOMIC must be specified when a statement immediately after **ATOMIC** is a macro operation such as summation or product.

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

Example:

```
      SUBROUTINE SUB(SUM, A, N)
      INTEGER::N
      REAL(KIND=8)::A(N,N), SUM
      ...
!NEC$ PARALLEL DO
      DO J = 1, N
         DO I = 1, N
!NEC$ ATOMIC
            SUM = SUM + A(I, J)
         ENDDO
      ENDDO
      ...
      END
```

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

```
FUNCTION FUN(N, A)
INTEGER N, I, J
REAL A(N), B(N)
REAL FUN
FUN = 1.0
...
!$OMP PARALLEL DO REDUCTION(+:FUN) ! OpenMP directive
DO J = 1, N
DO I = 1, N
FUN = A(J) + B(I) + FUN
END DO
END DO
RETURN
END FUNCTION FUN
```

7.2.2 OpenMP 5.0

The following features of OpenMP 5.0 are supported.

- **LOOP** construct
- **PARALLEL LOOP** construct
- **PARALLEL MASTER** construct

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 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**”.
- **PARALLEL DO SIMD** construct and **DO SIMD** construct
  Treated as **PARALLEL DO** and **DO** respectively**SIMD** construct
  Treated as **ivdep** directive
- **TASKLOOP** constructs
- **SIMD** construct
  If **SAFELEN** 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

7.2.5 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

```
FUNCTION FUN(N, A)
INTEGER N, I, J
REAL A(N), B(N)
REAL FUN
FUN = 1.0
...
!$OMP PARALLEL DO REDUCTION(+:FUN) ! OpenMP directive
DO J = 1, N
DO I = 1, N
FUN = A(J) + B(I) + FUN
END DO
END DO
RETURN
END FUNCTION FUN
```

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.

```
SUBROUTINE OMP_SET_NUM_THREADS(num_threads)    // Set number of threads
INTEGER num_threads
INTEGER FUNCTION OMP_GET_NUM_THREADS ()   // Get number of threads
INTEGER FUNCTION OMP_GET_MAX_THREADS()    // Get upper bounds on number of threads
INTEGER FUNCTION OMP_GET_THREAD_NUM()     // 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.

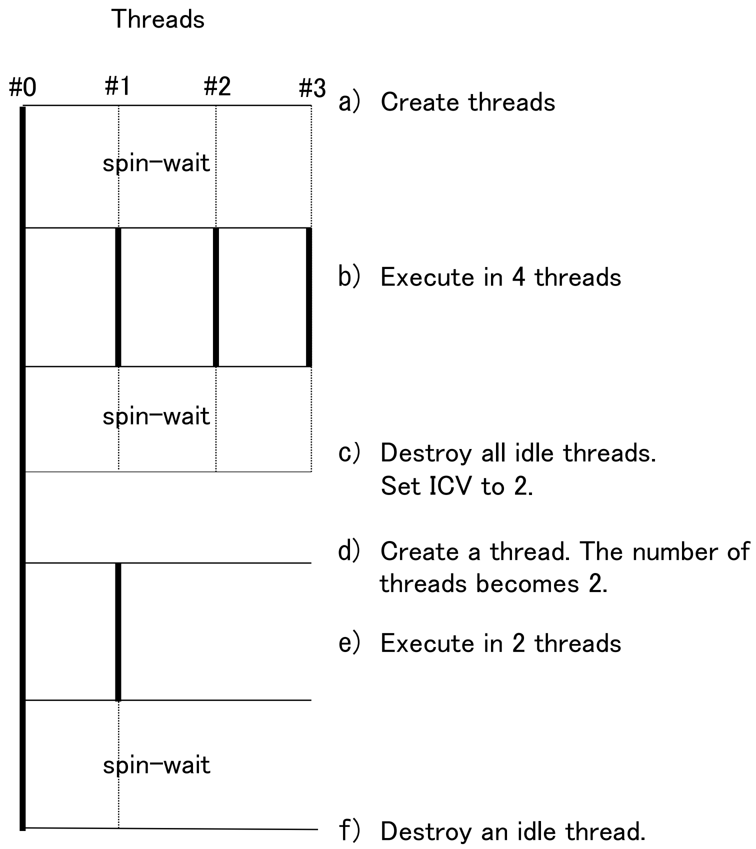
7.3.2 Thread Creation and Destroy

In automatic parallelized and OpenMP parallelized programs, the threads are created before the routine main program, 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.

```
Set OMP_NUM_THREADS=4
```

```
PROGRAM MAIN
...
!$OMP PARALLEL
   ...
   ...
!$OMP END PARALLEL
   ...
CALL OMP_SET_NUM_THREADS(2)
   ...
!$OMP PARALLEL
   ...
   ...
!$OMP END PARALLEL
   ...
END PROGRAM MAIN
```



(a) Three idle threads are created by master thread (#0) before main program 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.

7.3.3 Postpone Thread Creation

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

```
$ nfort -fopenmp -mno-create-threads-at-startup -static-nec a.o
$ nfort -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 procedure that includes procedure calls, the inside of the called procedure 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**.
- Threads for parallelization are created for each MPI process when a program is a MPI program. When a program uses 2 MPI processes and OMP_NUM_THREADS is set as 4, the program requires 8 cores (= 2 MPI * 4 threads) . When executing MPI program on VE, be careful not to run out of cores for execution.
- When outputting execution analysis information an auto-parallelized program using PROGINF and FTRACE, keep the following in check. See the manual "PROGINF/FTRACE User's Guide" for the detail of PROGINF or FTRACE.
  - The number of operations for the spin-waiting of the thread created before main program starts is included 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.

Chapter8　Compiler Listing

This chapter describes the output lists of the Fortran 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 Fortran Compiler (3.0.7) for Vector Engine   Thu Jun 18 13:25:29 2020     (a)

FILE NAME: fft.f90     (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 Fortran Compiler (1.0.0) for Vector Engine   Wed Jan 17 14:58:49 2018     (a)

FILE NAME: fft.f90        (b)

PROCEDURE 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 routine is not output in a diagnostic list for a routine that calls the inlined routine. Refer to the diagnostic list for the inlined routine 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 procedure together with the following information are output to the list.

- The vectorized status of loops and array expressions.
- The parallelized status of loops and array expressions.
- The status of inline expansion

8.3.1 Format of Format List

The format of the format list is as follows.

```
NEC Fortran Compiler (1.0.0) for Vector Engine   Wed Jan 17 15:00:01 2018   (a)
FILE NAME: a.f90     (b)

PROCEDURE NAME: SUB    (c)
FORMAT LIST

LINE   LOOP      STATEMENT
  (d)   (e)        (f)
     1:             SUBROUTINE SUB(A, B, N, M)
     2:             INTEGER::N, M
     3:             REAL(KIND=8)::A(M, N), B(M, N)
     4: +------>    DO J=1,M
     5: |V----->       DO I=1, N
     6: ||                A(I,J) = A(I,J) + B(I,J)
     7: |V-----        ENDDO
     8: +------     ENDDO
     9:             END SUBROUTINE
```

(a) Compiler revision and compilation date

(b) Name of source file

(c) Name of procedure

(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------> DO I = 1, N
|
V------  END DO
```

- The loop is partially vectorized.

```
S------>  DO I = 1, N
|
S------   END DO
```

- The loop is conditionally vectorized.

```
C------>  DO I = 1, N
|
C------   END DO
```

- The loop is parallelized.

```
P------>  DO I = 1, N
|
P------   END DO
```

- The loop is parallelized and vectorized.

```
Y------>  DO I = 1, N
|
Y------   END DO
```

- The loop is not vectorized

```
+------>  DO I = 1, N
|
+------   END DO
```

- The array expression is vectorized.

```
V======>  A = B + C
```

The sign "=" indicates that the beginning and the end of the loop exist in the same line.

- The nested loops are collapsed and vectorized.

```
W------>  DO I = 1, N
|*----->      DO J = 1, M
||
|*-----       END DO
W------   END DO
```

- The nested loops are interchanged and vectorized.

```
X------>  DO I = 1, N
```

```
|*----->       DO J
||
|*-----        END DO
X------   END DO
```

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

```
U------>  DO I = 1, N
|V----->       DO J
||
|V-----        END DO
U------   END DO
```

- The loops are fused and vectorized.

```
V------>  DO I = 1, N
|
|          END DO
|          DO I = 1, N
|
V------   END DO
```

- The loop is expanded.

```
*------>  DO I = 1, 4
|
*------   END DO
```

- 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

- Internal subprogram is output in the program unit which includes the subprogram.
- The loop structure or vectorization / parallelization status may be inexactly displayed when a part of the loop is included in a file which included by **INCLUDE** line or **#include**.
- The loop structure or vectorization / parallelization status may be inexactly displayed when two or more loops are written in a line.

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 Fortran Compiler (3.1.0) for Vector Engine     Thu Sep 17 07:33:16 2020   (a)

FILE NAME: fft.f90       (b)

FUNCTION NAME: func3         (c)

INLINE LIST

    INLINE REPORT: func3 (fft.f90:17)
    (d)
    -> INLINE: func2 (fft.f90:19)                    (e)
       -> NOINLINE: func0 (fft.f90:12)               (e)
          *** Source for routine not found.          (f)
       -> INLINE: func1 (fft.f90:13)                 (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++ Fortran (3.1.0) for Vector Engine    Thu Sep 17 08:10:39 2020   (a)

FILE NAME: vec.f90         (b)

FUNCTION NAME: func        (c)

VECTORIZATION LIST

  LOOP BEGIN: (vec.f90:3)
    <Unvectorized loop.>            (d)

    LOOP BEGIN: (vec.f90:4)
      <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 Fortran Compiler (3.1.0) for Vector Engine     Thu Sep 17 08:10:39 2020 (a)

FILE NAME: vec.f90       (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.f90:3)
   LOOP BEGIN: (vec.f90:4)
     *** 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 Non-Standard Extended Features

9.1.1 Statements

9.1.1.1 COMMON Statement

The Fortran compiler permits the mixing of character and other types of elements in the same common block. However this should be avoided if possible, because this may lower execution speed.

9.1.1.2 COMPLEX DOUBLE / COMPLEX DOUBLE PRECISION Statement

The **COMPLEX DOUBLE** / **COMPLEX DOUBLE PRECISION** statement, a type declaration statement provided for compatibility, specifies that all data entities whose names are declared in this statement are of intrinsic double precision complex type.

The kind parameter is "KIND(0.0D0)".

FORMAT

COMPLEX DOUBLE *entity-declaration-list*

COMPLEX DOUBLE PRECISION *entity-declaration-list*

where,

entity-declaration :

object-name [(*explicit-shape-spec*)][/ *initial-value* /]

| *object-name* [(*assumed-size-spec*)][/ *initial-value* /]

| *function-name*

9.1.1.3 COMPLEX QUADRUPLE / COMPLEX QUADRUPLE PRECISION Statement

The **COMPLEX QUADRUPLE** / **COMPLEX QUADRUPLE PRECISION** statement provided for compatibility, a type declaration statement, specifies that all data entities whose names are declared in this statement are of intrinsic quadruple precision complex type.

The kind parameter is "KIND(0.0Q0)".

FORMAT

COMPLEX QUADRUPLE *entity-declaration-list*

COMPLEX QUADRUPLE PRECISION *entity-declaration-list*

where,

entity-declaration :

object-name [(*explicit-shape-spec*)] [/ *initial-value* /]

| *object-name* [(*assumed-size-spec*)] [/ *initial-value* /]

| *function-name*

9.1.1.4 DATA Statement

The Fortran compiler permits writing a Hollerith constant, the number of characters is more than 4, to the initial value of a **DATA** statement.

9.1.1.5 DIMENSION Statement

An initial value can be set in the **DIMENSION** statement in the same way as in the **DATA** statement and a type declaration statement.

FORMAT

DIMENSION *array-name*(*array-shape-spec*) [/ *init-val-expr-list* /]

[,*array-name*(*array-shape-spec*)[/ *init-val-expr-list* /]]

...

where the *init-val-expr-list* represents the initial value of the immediately preceding array name.

The rules to set the initial value are the same as those of the **DATA** statement.

9.1.1.6 DOUBLE Statement

The **DOUBLE** statement, a type declaration statement provided for compatibility, specifies that all data entities whose names are declared in this statement are of intrinsic double precision real type.

The kind parameter is "KIND(0.0D0)".

FORMAT

DOUBLE *entity-declaration-list*

where,

entity-declaration :

object-name [(*explicit-shape-spec*)] [/ *initial-value* /]

| *object-name* [(*assumed-size-spec*)] [/ *initial-value* /]

| *function-name*

9.1.1.7 DOUBLE COMPLEX Statement

The **DOUBLE COMPLEX** statement, a type declaration statement provided for compatibility, specifies that all data entities whose names are declared in this statement are of intrinsic double precision complex type.

The kind parameter is "KIND(0.0D0)".

FORMAT

DOUBLE COMPLEX *entity-declaration-list*

where,

entity-declaration :

object-name [(*explicit-shape-spec*)] [/ *initial-value* /]

| *object-name* [(*assumed-size-spec*)] [/ *initial-value* /]

| *function-name*

9.1.1.8 DOUBLE PRECISION Statement

Initial values can be specified for the entities whose names are declared in the **DOUBLE PRECISION** statement.

FORMAT

DOUBLE PRECISION [[,*attribute-spec*]... ::] *entity-declaration-list*

where,

attribute-spec :

ALLOCATABLE

| DIMENSION(*array-spec*)

| EXTERNAL

| INTENT(*intent-spec*)

| INTRINSIC

| OPTIONAL

| PARAMETER

| POINTER

| PRIVATE

| PUBLIC

| SAVE

| TARGET

entity-declaration :

object-name [(*explicit-shape-spec*)] [/ *initial-value* /]

| *object-name* [(*assumed-size-spec*)] [/ *initial-value* /]

| *function-name*

9.1.1.9 EQUIVALENCE Statement

The Fortran compiler permits the association of character-type elements with other types (without a derived type). However, this should be avoided, to maintain compatibility with other implementations of Fortran.

9.1.1.10 FORMAT Statement

The Fortran compiler permits the comma separator to be omitted immediately before and after character string edit descriptors in **FORMAT** statements. Note, however, that the comma separator between the X edit descriptor and the character string edit descriptor must not be omitted.

Furthermore, the compiler permits *n* in *n*X edit descriptor and *k* in *k*P edit descriptor to be omitted. When it is omitted, the default value is one. The data edit descriptor (B/D/E/EN/ES/F/G/I/L/O/Z) can be specified only the edit descriptor.

Example:

```
          PRINT 10, 3.14, 2.71
          PRINT 20, 3.14, 2.7110
          FORMAT('PI='F4.2' and',X,'E='F4.2)
20        FORMAT('PI='F' and',X,'E='F)
```

This produces the output:

```
PI=3.14 and E=2.71
PI=        3.1400001 and E=        2.7100000
```

9.1.1.11 FUNCTION Statement

A string "([*dummy-argument-name-list*])" following a function-name can be omitted including "( )".

In this case, the format of the **FUNCTION** statement is as follows:

FORMAT

[*type-spec*] FUNCTION *func-name* [([*dummy-arg-name-list*])]

where,

type-spec :

INTEGER [**byte-count*]

| REAL [**byte-count*]

| DOUBLE PRECISION

| DOUBLE

| QUARUPLE PRECISION

| QUADRUPLE

| COMPLEX [**byte-count*]

| COMPLEX DOUBLE PRECISION

| COMPLEX DOUBLE

| DOUBLE COMPLEX

| COMPLEX QUADRUPLE PRECISION

| COMPLEX QUADRUPLE

| LOGICAL [**byte-count*]

[*type-spec*] FUNCTION *func-name* [([*dummy-arg-name-list*])]

where,

type-spec :

CHARACTER [**character-length*]

9.1.1.12 Computed GO TO Statement

The following computed **GO TO** statement is available.

FORMAT

GO TO (*statement-label-list*) [,] *scalar-integer-expr*

SYNTAX RULE

Each statement-label within the statement-label-list must be the statement-label of a branch target statement within the same scoping unit as the computed **GO TO** statement.

GENERAL RULE

- The same statement-label may be written more than once within a single *statement-label-list*.
- When a computed **GO TO** statement is executed, the *scalar-integer-expr* is evaluated. Assume this value is i and the number of statement-labels within the *statement-label-list* is n. If 1 <= i <= n, a transfer of control occurs, and the statement having the i-th statement-label within the *statement-label-list* is executed next. If i < 1 or i > n, the execution sequence continues as though a **CONTINUE** statement were executed.

Example:

```
GO TO (100, 200, 300, 400, 500), I
```

9.1.1.13 Arithmetic IF Statement

The following arithmetic **IF** statement is available.

FORMAT

IF (*scalar-numeric-expr*) *stmt-label*, *stmt-label*, *stmt-label*

SYNTAX RULE

- Each *stmt-label* must be the statement-label of a branch target statement within the same scoping unit as the arithmetic **IF** statement.
- The scalar-numeric-expr must not be of complex type.
- A maximum of two *stmt-labels* may be omitted; however, the comma must not be omitted. If the *stmt-label* corresponds to the scalar-numeric-expr, the execution sequence continues as if the **CONTINUE** statement were executed.
- An arithmetic **IF** statement in which at least one of the *stmt-labels* is omitted can be used as a terminal statement of a **DO** loop.

GENERAL RULE

- The same *stmt-label* can be written more than once within a single arithmetic **IF** statement.
- If an arithmetic **IF** statement is executed, a *scalar-numeric-expr* is evaluated, followed by a transfer of control. The branch target expression identified by the first, second, or third statement-label is executed next according to whether the value of the *scalar-numeric-expression* is negative, zero, or positive.

Example:

```
IF( I + J ) 100, 200, 300
```

9.1.1.14 IMPLICIT Statement

The same letter may be specified more than once, either written as an individual letter or included in a range of letters indicated by a letter-specification, throughout all **IMPLICIT** statements in a single scoping unit. If the same letter is specified more than once, the last letter is effective.

An **IMPLICIT** statement can implicitly specify the type and type parameters of a data entity whose name starts with "$".

9.1.1.15 PARAMETER Statement

In **PARAMETER** statement, "( )" in the list can be omitted. When omitting, the constant form, not the implicit typing of the name, determines the data type of the variable.

Example:

```
PARAMETER PI=3.1415927, DPI=3.141592653589793238D0
PARAMETER PIOV2=PI/2, DPIOV2=DPI/2
PARAMETER FLAG=.TRUE., LONGNAME='A STRING OF 25 CHARACTERS'
PRINT *,'PI=',PI
PRINT *,'DPI=',DPI
PRINT *,'PIOV2=',PIOV2
PRINT *,'DPIOV2=',DPIOV2
PRINT *,'FLAG=',FLAG
PRINT *,'LONGNAME=',LONGNAME
END
```

This produces the output:

```
PI=   3.1415927
DPI=   3.1415926535897931
PIOV2=   1.5707964
DPIOV2=   1.5707963267948966
FLAG= T
LONGNAME=A STRING OF 25 CHARACTERS
```

9.1.1.16 FORTRAN77 POINTER Statement

The following **POINTER** statement provided for compatibility is available.

FORMAT

POINTER (*pointer-variable*, *data-variable-declaration*) [,(*pointer-variable*, *data-variable-declaration*)]...

where,

pointer-variable :

scalar-8byte-integer-variable

data-variable-declaration :

scalar-variable-name

| *array-name*

| *array-name* (*explicit-shape-specification*)

| *array-name* (*assumed-shape-specification*)

GENERAL RULE

- A FORTRAN77 **POINTER** statement cannot appear in a module specification part.

- A pointer-variable must be a scalar variable.
- A pointer-variable must not have the **ALLOCATBLE** attribute.
- A pointer-variable must be declared as of type 8-byte integer.
- A pointer-variable must not have the **POINTER** or **TARGET** attribute.
- A pointer-variable must not be a component of a derived type.
- A pointer-variable cannot appear in a **PARAMETER** statement or in a type declaration statement that includes the **PARAMETER** attribute.
- A pointer-variable cannot appear in a **DATA** statement.
- A data-variable-declaration must not be an assumed-shape array.
- A data-variable-declaration must not have the **ALLOCATBLE**, **INTENT**, **OPTIONAL**, **DUMMY**, **TARGET**, **INTRINSIC** or **POINTER** attribute.
- A data-variable-declaration cannot appear in two or more **POINTER** statements.
- A data-variable-declaration must not be a pointer-variable.
- If data-variable-declaration is an array specification, it must be explicit-shape or assumed-size.
- A data-variable-declaration cannot appear in a **SAVE**, **DATA**, **EQUIVALENCE**, **COMMON** or **PARAMETER** statement.
- A data-variable-declaration must not be of a derived type or be a component of a derived type.
- A data-variable-declaration must be of an intrinsic type.
- A data-variable-declaration must not be a name of a common block object, a dummy argument, a function result or an automatic data object.N

NOTE

- A pointer-variable is processed the same way as an ordinary variable of type 8-byte integer.
- If the explicit declaration of the pointer-variable type is omitted, the type is determined implicitly as 8-byte integer.
- A pointer-variable can be declared for one or more data-variable-declarations.

- If a data-variable-declaration is an array specification and its upper and lower bounds are not constant, the size of the array is determined at entry to the procedure.
- A storage unit for a data-variable-declaration is not allocated. The actual address of it is dynamically determined by specifying the value of the corresponding pointer-variable as byte-address.
- If a data-variable-declaration is an array, its shape can be determined by a declaration statement, a **DIMENSION** statement or a **POINTER** statement.
- A pointer-variable cannot be accessed by host association.
- A FORTRAN77 **POINTER** statement can appear in a block data program unit.

9.1.1.17 QUADRUPLE / QUADRUPLE PRECISION Statement

The **QUADRUPLE** / **QUADRUPLE PRECISION** statement provided for compatibility, a type declaration statement, specifies that all data entities whose names are declared in this statement are of intrinsic quadruple precision real type. The kind parameter is "KIND(0.0Q0)".

FORMAT

COMPLEX QUADRUPLE *entity-declaration-list*

COMPLEX QUADRUPLE PRECISION *entity-declaration-list*

where,

entry-declaration :

object-name [(*explicit-shape-spec*)] [/*initial-value*/]

| *object-name* [(*assumed-size-spec*)] [/*initial-value*/]

| *function-name*

9.1.1.18 RETURN Statement

A real type expression can be specified in a scalar integer expression of the **RETURN** statement.

The specified real type expression is converted to the integer type prior to control transfer.

9.1.1.19 STOP Statement

A scalar variable name or constant name of character type or default integer type can be specified as the stop-code.

9.1.2 Program

9.1.2.1 Statement Continuation

The maximum number of continuation lines is 511 lines in any source forms.

9.1.2.2 Currency Symbol $

The currency symbol ($) can be used in place of a letter in a name.

The currency symbol ($) can be also used for an edit descriptor in a formatted record. This specifies the suppression, on output, of vertical spacing control for the last record of the format control. If a $ edit descriptor is specified on input, it is ignored.

9.1.2.3 Argument Association

A procedure without an explicit interface can be normally compiled even if it has the following arguments which violate the standard rules governing argument association.

- The number of the actual arguments is less than the number of the dummy arguments.
- An argument is of type character, and the length of the dummy argument is greater than the length of the actual argument.

9.1.2.4 Array Complement

When the rank of an array is specified lower than its declaration, the compiler complements the lower bounds of the omitted ranks.

Example:

Declare	Reference	Reference after complement
A(2,3)	A(1)	A(1,1)
B(2,-4:4)	B(1)	B(1,-4)
C(2,3,4,5)	C(2)	C(2,1,1,1)

9.1.3 Source Form

9.1.3.1 Fixed Source Form

- Statement Continuation
  For compatibility, if "&" is specified in character position 1, all subsequent characters of that line beginning with character position 2 constitute the continuation line of the preceding line that is not a comment.
- Extended Fixed Source Form
  Maximum length of one line is 2,048 characters. This form is the same as the fixed source form except that a line is not fixed on 72 columns, but a line length is variable up to 2,048 columns.
  In the extended fixed source form, a statement can consist of up to 13,200 characters including an initial line.
  In the standard Fortran, the maximum number of continuation lines is 255 lines in any source forms.
  When **-fextend-source** is specified, the extended fixed source form is enabled.
- Tab Code Line
  When the first tab code appears in character positions 1 through 6, if the character following the first tab code is a digit, that character is considered to have appeared in character position 6; if the character following the first tab code is not a digit, that character is considered to have appeared in character position 7. In this case, everything up to the last character of the line becomes a portion of the statement. Also, if the first tab code appears in character position 7 or after, it is considered to be blank except in a character constant, Hollerith constant, or character string edit descriptor.

9.1.3.2 Free Source Form

In free source form, there is no limit for the maximum length of one line.

9.1.4 Expressions

9.1.4.1 Relational Operator

For compatibility, the following relational-operators can be used:

=>

| =<

| ><

| <>

9.1.4.2 Logical Operator

For compatibility, the following logical operator can be used:

.XOR.

9.1.4.3 Maximum Array Rank

The maximum rank of an array is 31. The Fortran 2008 standard only requires 15, and previous Fortran standard only required 7.

9.1.4.4 Boz-literal-constant

A boz-literal-constant in the format containing a quotation mark or an apostrophe may be specified as the following too.

- An initialization value of a **PARAMETER** statement.
- An initialization value of a type declaration statement.
- An actual argument of a procedure having an implicit interface.

Then the type of a boz-literal-constant is fixed by its usage. When the length of the boz-literal-constant is less than the length of the type, the leftmost digits have a value of zero. When the length of the boz-literal-constant is more than the length of the type, the leftmost digits are truncated.

A hexadecimal-constant can also be written with "X" instead of "Z" in the format shown below:

X"hexadecimal-digit [*hexadecimal-digit*] ..."

| X'hexadecimal-digit [*hexadecimal-digit*] ...'

9.1.4.5 Hollerith Type

A Hollerith constant can be written only in a Hollerith relational expression and a Hollerith assignment statement.

- Hollerith Relational Expression

  If one operand is a Hollerith constant or character constant in a relational expression, the other operand may be a scalar variable of integer type or real type. This makes it possible to compare Hollerith data. The variable must be defined with Hollerith data at the time of evaluation of the relational expression. The Hollerith relational expression is interpreted in the same manner as a character expression having the same character value.

  Example:

```
INTEGER DATA
READ(*, 10) DATA
10  FORMAT(A4)
IF( DATA .EQ. 3HEND ) STOP
```

- Hollerith Assignment Statement

  In a Hollerith assignment statement, if the right side is a Hollerith constant or character constant, the left side may be any non-character type scalar variable. The execution of this assignment statement defines the variable on the left side with the Hollerith data on the right side.

  Assume n as the number of characters in a Hollerith constant or a character constant, and assume g as the number of characters that can be contained in the variable on the left side. If n is not greater than g, g characters are assigned by extending the right side of the constant with g-n blank characters. If g is not greater than n, the g characters on the left side of the constant are assigned.

  Example:

```
INTEGER TITLE
TITLE = 4HDATA
WRITE(*, 10) TITLE
10 FORMAT(A4)
```

9.1.4.6 Subscript Expression and Substring Expression

A real type expression can be specified in the subscript expression or substring expression in an array element.

The specified real type expression is converted into integer type prior to calculating the subscript value.

9.1.5 Deleted Features

The Fortran compiler supports the deleted features in Fortran95 (**PAUSE** statement, **ASSIGN** statement, assigned **GO TO** statement, and H edit descriptor). When **-Wobsolescent** is valid and these features are found, a warning message with "Deleted feature:" is output.

9.2 Implementation-Defined Specifications

9.2.1 Data Types

9.2.1.1 Correspondence Between Kind Type Parameters and Data Types

The available kind values and correspondence between kind type parameters and data types are as follows.

Type	Kind Type Parameter	Data Type
integer	1	1-byte integer
integer	2	2-byte integer
integer	4	4-byte integer (default integer type)
integer	8	8-byte integer
real	4	4-byte real (default real type)
real	8	8-byte real
real	16	16-byte real
complex	4	(4,4)-byte complex (default complex type)
complex	8	(8,8)-byte complex
complex	16	(16,16)-byte complex
logical	1	1-byte logical
logical	4	4-byte logical (default logical type)
logical	8	8-byte logical
character	1	character (default character type)

9.2.2 Internal Representation of Data

9.2.2.1 Integer Type

An integer data item has 1, 2, 4, or 8 consecutive bytes in a memory sequence. It is stored in binary form, with the rightmost bit position representing the digit 1. A negative number is represented by 2's complement notation. The leftmost bit is the sign; 0 is positive, 1 is negative.

- 1-byte Integer

  SYNOPSIS

7	0
S	

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

EXPRESSIBLE VALUE

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

- 2-byte Integer

SYNOPSIS

15	0
S	

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

EXPRESSIBLE VALUE

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

- 4-byte Integer

SYNOPSIS

31	0
S	

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

EXPRESSIBLE VALUE

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

- 8-byte Integer

SYNOPSIS

63	0
S	

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

EXPRESSIBLE VALUE

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

9.2.2.2 Floating-Point Data

- Real Type

A real data item occupies 4 consecutive bytes in a memory area. The leftmost bit is the sign bit of the mantissa. The 23 bits on the right are the mantissa. The mantissa is stored in binary representation, with its leftmost bit being the 2^{-1} place. When the sign bit of the mantissa in the leftmost bit position is 0, the mantissa is a positive value. When it is 1, the mantissa is the absolute value of a negative number. The 8 bits following the leftmost bit are the exponent. The exponent is stored in binary representation, with its leftmost bit being the unit's place. The value 0 is represented by making the value of the exponent 0.

SYNOPSIS

31	23	22 0
S	E	M

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

(A head bit of M is 0:signaling NaN, A head bit of M is 1:quiet NaN)

Infinity E == 255 and M == 0

Signed Zero E == 0

- Double-Precision Type

A double-precision real data item occupies 8 consecutive bytes in a memory area. The leftmost bit is the sign bit of the mantissa. The 52 bits on the right are the mantissa. The mantissa is stored in binary representation, with its leftmost bit being the 2^{-1} place. When the sign bit of the mantissa in the leftmost bit position is 0, the mantissa is a positive value. When it is 1, the mantissa is the absolute value of a negative number. The 11 bits following the leftmost bit are the exponent. The exponent is stored in binary representation, with its leftmost bit being the unit's place. The value 0 is represented by making the value of the exponent 0.

SYNOPSIS

63	52	51 0
S	E	M

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

(A head bit of M is 0:signaling NaN, A head bit of M is 1:quiet NaN)

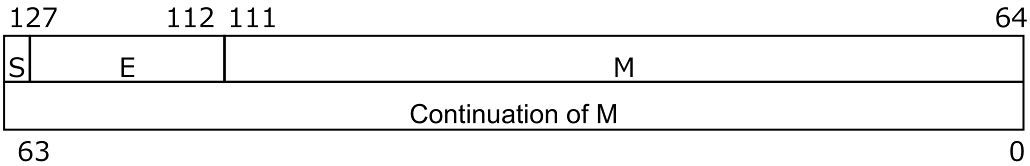
Infinity　　　E == 2047 and M == 0

Signed Zero　E == 0

- Quadruple-Precision Type

A quadruple-precision real data item occupies 16 consecutive bytes in a memory area. The leftmost bit is the sign bit of the mantissa. The 112 bits on the right are the mantissa. The mantissa is stored in binary representation, with its leftmost bit being the 2^{-1} place. When the sign bit of the mantissa in the leftmost bit position is 0, the mantissa is a positive value. When it is 1, the mantissa is the absolute value of a negative number. The 15 bits following the leftmost bit are the exponent. The exponent is stored in binary representation, with its leftmost bit being the unit's place. The value 0 is represented by making the value of the exponent 0.

SYNOPSIS



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

E: Exponent (0<=E<=32767)

M: Mantissa (0<=M<1)

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

9.2.2.3 Complex Type

- Complex Single-Precision Type

  A single-precision complex data item occupies 8 consecutive bytes in a memory area. The 4 bytes occupying the low-order addresses store the real part, and the 4 bytes occupying the high-order addresses store the imaginary part. The real and imaginary parts are in the same format as real data.

  SYNOPSIS

63	55	54 32
RS	RE	RM
IS	IE	IM
31	23	22 0

  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 10^{37}.

  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

- Complex Double-Precision Type

  A double-precision complex data item occupies 16 consecutive bytes in a memory area. The 8 bytes occupying the low-order addresses store the real part, and the 8 bytes occupying the high-order addresses store the imaginary part. The real and imaginary parts are in the same format as double-precision real data.

  SYNOPSIS

127	116	115 64
RS	RE	RM
IS	IE	IM
63	52	51 0

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 10^{308}.

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

- Complex Quadruple-Precision Type

  A quadruple-precision complex data item occupies 32 consecutive bytes in a memory area. The 16 bytes occupying the low-order addresses store the real part, and the 16 bytes occupying the high-order addresses store the imaginary part. The real and imaginary parts are in the same format as quadruple-precision real data.

SYNOPSIS

255	240	239 192
RS	RE	RM
	Continuation of M	
	IE	IM
	Continuation of M	
63		0

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

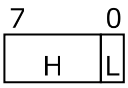
Signed Zero RE == 0 and IE == 0

9.2.2.4 Logical Type

A logical data item has 1 byte, 4 consecutive bytes, or 8 consecutive bytes in a memory sequence.

- 1-byte Logical

SYNOPSIS

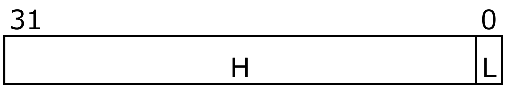


L: The lowest bit (0: False, 1: True)

H: Higher bit (H==0)

- 4-byte Logical

SYNOPSIS

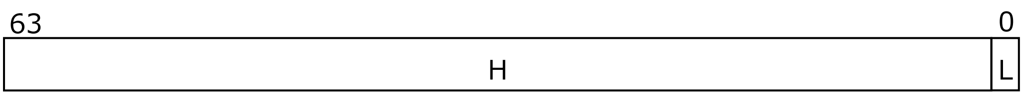


L: The lowest bit (0: False, 1: True)

H: Higher bit (H==0)

- 8-byte Logical

SYNOPSIS



L: The lowest bit (0: False, 1: True)

H: Higher bit (H==0)

9.2.2.5 Character Type

A character data item occupies as many contiguous bytes of memory as specified by a type or **IMPLICIT** statement. If the item is a character constant, it occupies as many contiguous bytes as its number of characters.

SYNOPSIS

BYTE	1	2	3	4		n-1	n
	C_1	C_2	C_3	C_4		C_{n-1}	C_n

Ci: i-th character from the left

n: Length of a character-type scalar variable or array element specified by a type or **IMPLICIT** statement (up to 32767 characters), or the length of a character constant (up to 16383 characters)

9.2.2.6 Hollerith Type

An item of Hollerith data occupies contiguous 1, 2, 4, 8, 16, or 32 bytes of memory and is left-justified when stored. It is stored in a variable or array element of a type other than character type, followed by the necessary number of blanks.

A Hollerith constant consists of an unsigned nonzero integer n, the following letter H and the following string of n consecutive characters. This string may consist of any characters capable of representation in the processor. The string of n characters is Hollerith data.

The following example shows 5HABCDE stored in a variable of double-precision floating-point format 1 data.

BYTE	1	2	3	4	5	6	7	8
	A	B	C	D	E	[ ]	[ ]	[ ]

"[ ]" indicates blank

A Hollerith constant can be written only in a Hollerith relational expression, a Hollerith assignment statement, type-statement in FORTRAN77 compatible format, a **DATA** statement, a **DIMENSION** statement, or an actual argument list in a procedure reference having no explicit interface.

9.2.2.7 Hexadecimal Type

An item of hexadecimal data is stored according to an initial value setting in a **DATA** or type, or by executing a **READ** statement using a Z edit descriptor. It occupies as many bytes of memory as required for the type of data, and is left-justified when stored. One byte of hexadecimal data contains two hexadecimal digits. Each hexadecimal digit is represented by 4 bits.

9.2.2.8 Octal Type

An item of octal data is stored according to an initial value setting in a **DATA** or type statement, or by executing a **READ** statement using an O edit descriptor. It occupies as many bytes of memory as required for the type of data, and is left-justified when stored. Three bits represent one octal digit.

9.2.2.9 Binary Type

An item of binary data is stored according to an initial value setting in a **DATA** or type statement, or by executing a **READ** statement using a B edit descriptor. It occupies as many bytes of memory as required for the type of data, and is left-justified when stored. One bit represents one digit of binary data.

9.2.2.10 Special Values

Floating-point data can be used for the following special values:

- Nonnumeral (NaN)

  A nonnumeral indicates that numeric representation cannot be used as a result of an invalid operation. For example, the result of the operation "0.0/0.0" is a nonnumeral.

  Nonnumerals are classified into the following two types.

  - Signaling NaN

    If this type of nonnumeral is used for an operation, an invalid operation exception is detected.

  - Quiet NaN

    Quiet NaN: This type of nonnumeral is returned as the result of an invalid operation. However, no invalid operation exception is detected.

- Infinite (inf)

  Infinities are classified into the positive infinite and the negative infinite. The positive infinite (+inf) is the value that is greater than any other numeric values that can be represented in the same format as the positive infinite. The negative infinite (-inf) is the value that is less than any other numeric values that can be represented in the same format as the negative infinite.

- Signed zero (+0 and -0)

  In internal representation, +0 and -0 are distinguished from each other by sign.

However, these two values are treated as the same value.

```
0.0 .EQ. (-0.0) => true
```

As shown below, a signed 0 is effective in obtaining a positive or negative infinite value.

```
B1 = +0.0
B2 = -0.0
A1 = 1.0 / B1
A2 = 1.0 / B2
WRITE(*, *) "A1 = ", A1, " A2 = ", A2
```

9.2.3 Specifications

Various upper limits in the Fortran compiler are as described below.

Items	Upper Limits
Nesting level of files included by **INCLUDE** line	63
Rank of an array	31
Number of continuation lines	1023
Length of a name	199

9.2.4 Predefined Macro

All predefined macros are enabled when a source program is preprocessed by **fpp** and one of the following conditions is satisfied.

- **-E** or **-M** is specified.
- The suffix of input source file is **.F**, **.F90**, **.F95**, or **.F03**.

Predefined macros are as follows.

unix, __unix, __unix__

Always defined as 1.

linux, __linux, __linux__

Always defined as 1.

__gnu_linux__

Always defined as 1.

__ve, __ve__

Always defined as 1.

__VE_ARCH_1__

Always defined as 1.

__VE_ARCH_3__

Defined as 1 when **−march**=ve3 is enabled; Otherwise not defined.

__ELF__

Always defined as 1.

__FP16_FORMAT__

Sets the format of half-precision floating-point.

Defined as 1 when **−march**=ve3 and **-mfp16-format**=ieee are enabled;

Defined as 2 when **−march**=ve3 and **-mfp16-format**=bfloat are enabled;

Otherwise not defined.

__FP16_IEEE

Always defined as 1.

__FP16_BFLOAT

Always defined as 2.

__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 **-O***n* 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.

9.2.5 Notes for Intrinsic Procedures

CPU_TIME

Return CPU time for program execution. When parallelization in version 3.0.7 or later, this subroutine returns the CPU time of thread that called **CPU_TIME**. In previous versions, this subroutine returned accumulated CPU time of all threads. If you want to get accumulated CPU time of all threads in this version or later, specify "YES" in environment variable **VE_FORT_ACCUMULATE_THREAD_CPU_TIME**.

9.3 Memory Allocation and Deallocation

Fortran compiler has a memory block management feature to accelerate allocation and deallocation for memory which is allocated by **ALLOCATE** statement, deallocated by **DEALLOCATE** statement, and work area using in some statements.

By a memory block management feature, three memory blocks are reserved at the start of program execution, and a memory chunk in the blocks are assigned as a memory area for scalar variable (basic and derived types) and small size arrays. Therefore, system calls to allocate and deallocate memory chunks can be omitted for them.

9.3.1 Memory block

There are three types of memory blocks depending on the type of data to be allocated, and each has a size of 64 megabytes at the start of program execution. A data whose size is less than a threshold size is assigned in a memory block. The threshold size is 16 megabytes by default.

Block Type	Allocated Data Type	Size	Threshold Size
Allocate	Scalars and arrays having ALLOCATABLE attribute	64	16
Pointer	Scalars and arrays having POINTER attribute	64	16
Miscellaneous	Automatic arrays, work arrays and work area needed by compiler	64	16

(Unit: Megabyte)

A data whose size is greater than or equal to the threshold size is allocated or deallocated by malloc(3C) or free(3C) which is called from Fortran compiler’s runtime routine.

When a sufficient area for an allocated data cannot be found in a memory block, new memory block whose size is “Size” is added.

9.3.2 Change size and threshold size of memory block

The size of memory block can be changed by the environment variable **VE_FORT_MEM_BLOCKSIZE**. The value can be specified as megabytes by using “M” as unit and gigabytes by using “G” as unit. The value must be power of 2. The size is set 64 megabytes when it is not specified explicitly. The threshold size is set to “size”/4.

```
$ export VE_FORT_MEM_BLOCKSIZE=32M
```

The size is set to 32megabytes and the threshold size is set to 8 megabytes by the above setting.

9.4 Run-Time Input/Output

9.4.1 Formatted Records

Formatted records are input or output using a formatted, list-directed, or namelist input/output statement.

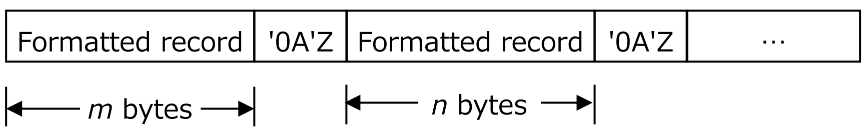
Records with a formatted input/output statement are input or output in accordance with the format specification. In general, this type of record has a variable length, but cannot be longer than the record buffer provided by the Fortran compiler.

Records with a list-directed input/output statement are input or output in accordance with the input/output list of that statement. When a list-directed input/output statement is executed once, one or more records are input or output.

Records with a namelist input/output statement are input or output in accordance with the specified list of namelist names. When a namelist input/output statement is executed once, one or more records may be input or output.

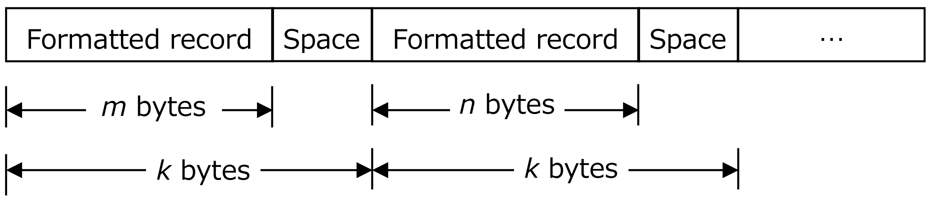
9.4.1.1 Sequential File Formatted Records

Sequential file formatted records are separated from each other by new line codes ('0A'Z). Each record has a variable length. The format is shown here.



9.4.1.2　Direct File Formatted Records

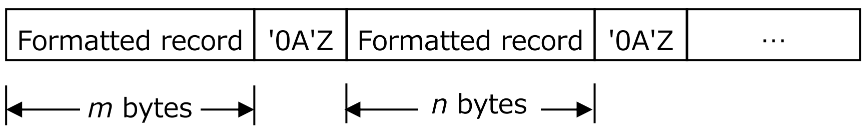
The length of a formatted record in a direct file is specified by the **RECL** specifier in an **OPEN** statement. When a record created by input/output list-item editing is shorter than the length of the records in a file, the record is padded with spaces to the right.



(*k*: Length specified by an **OPEN** statement)

9.4.1.3　Stream File Format Records

Stream file formatted records are separated from each other by new line codes ('0A'Z), same as sequential file formatted records. However, the maximum length of the records does not apply to this format. The format is shown here.

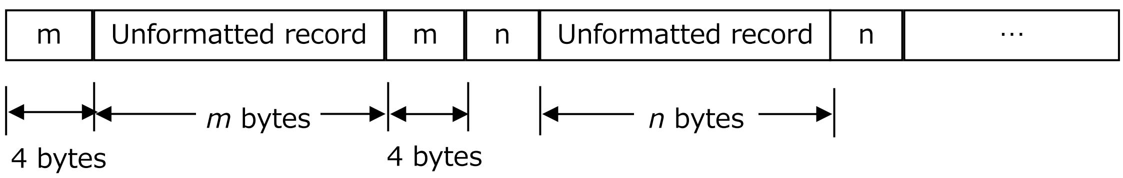


9.4.2　Unformatted Records

Unformatted records are input or output only with an unformatted input/output statement. The length of an unformatted record is the same as total data size of input/output items. Please refer to Section 7.2 about each data size.

9.4.2.1　Sequential File Unformatted Records

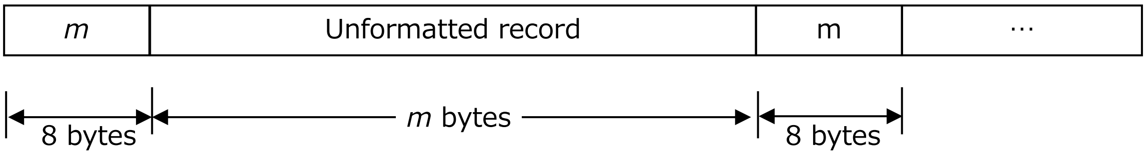
Each unformatted record in a sequential file is preceded and followed by 4-byte data that indicates the byte length of the record as shown in this example.



(*m*,*n*: Byte length of record)

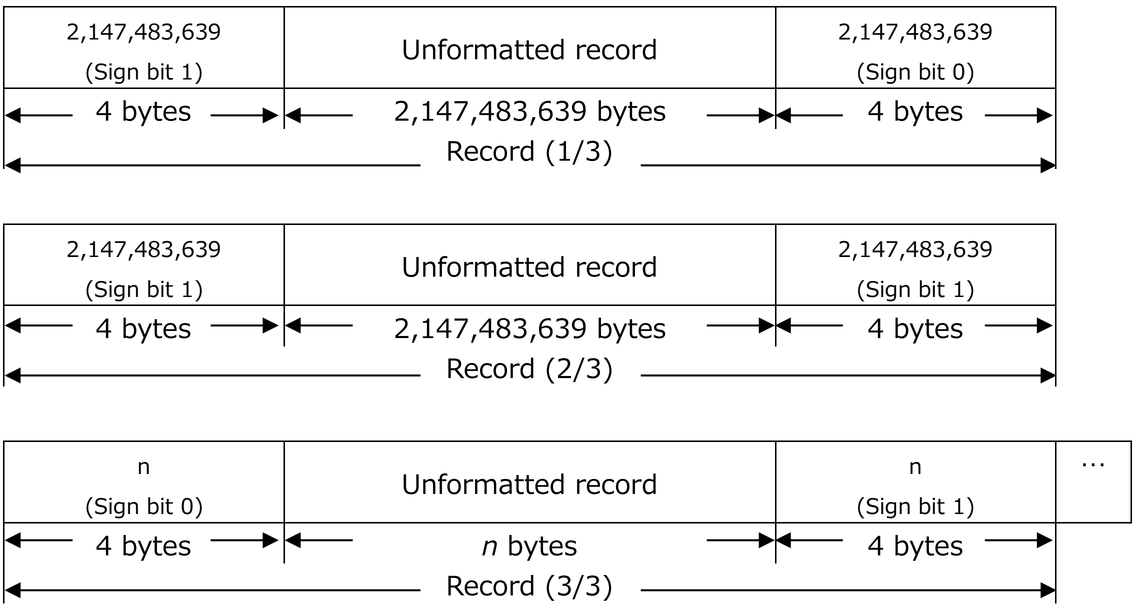
When the environment variable **VE_FORT_EXPRCW** is specified, each unformatted

record in a sequential file is preceded and followed by 8-byte data that indicates the byte length of the record as shown in this example.
This record format is able to handle the records over 2 giga bytes.



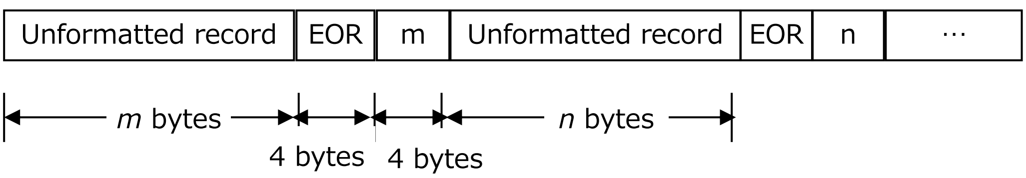
(*m*: Byte length of record)

When the environment variable **VE_FORT_SUBRCW** is specified, each unformatted record in a sequential file is divided into 2,147,483,639 bytes or less. This records are preceded and followed by 4-byte data that indicates the byte length of the record as shown in this example. The sign bit in this length field indicates whether the preceding and following records are continued.



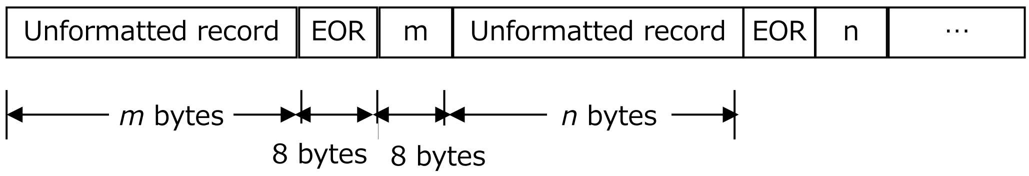
(2,147,483,639, *n*: Byte length of record)

When the environment variable **VE_FORT_PARTRCW** is specified, each unformatted record in a sequential file is followed by 4-byte data that indicates EOR and the byte length of the record as shown in this example.



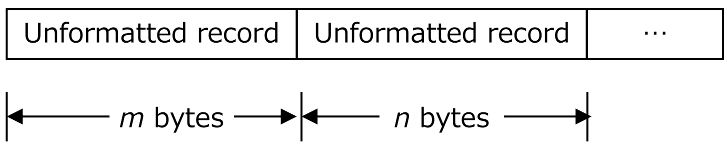
(*m*,*n*: Byte length of record)

When the runtime options **VE_FORT_EXPRCW** and **VE_FORT_PARTRCW** are specified at the same time, each unformatted record in a sequential file is followed by 8-byte data that indicates EOR and the byte length of the record as shown in this example.



(*m*,*n*: Byte length of record)

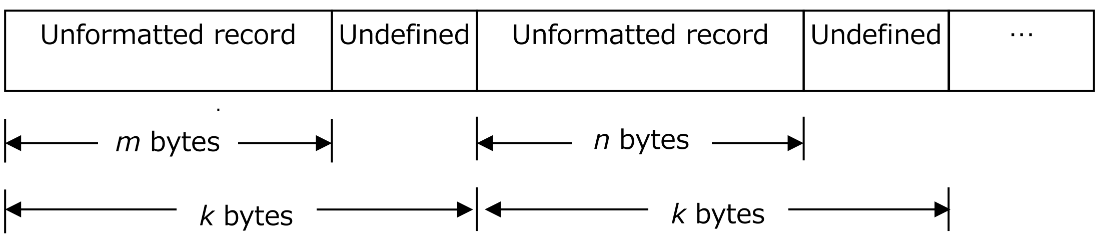
When the environment variable **VE_FORT_NORCW** is specified, each unformatted record in a sequential file is preceded and followed by no control record data as shown in this example. This is the same as unformatted record of stream file.



9.4.2.2 Direct File Unformatted Records

The length of an unformatted record in a direct file is specified by the **RECL** specifier in an **OPEN** statement. When a record consisting of input/output list items is shorter than the length of records in a file, the remainder of the record is undefined, as follows.

When writing an unformatted record to a file, the undefined data are ignored and the length of the record will be the same as the total data size of output items.



(*k*: Length specified by an **OPEN** statement)

9.4.2.3 Stream File Unformatted Records

An unformatted stream file is a byte stream without records.

Unformatted byte stream ....

9.4.3 Preconnection

An external unit identifier is defined to identify a specific file before program execution is started. This is called a preconnection.

9.4.3.1 System Standard File Preconnection

System standard files are preconnected to external unit identifiers as follows.

External Unit Identifier	System Standard File
0	Standard error output
5	Standard input file
6	Standard output file

A preconnection with an external unit identifier is valid until an **OPEN** statement is executed for the external unit identifier. Once an **OPEN** statement is executed, the external unit identifier is disconnected from the system standard file. Reconnection is impossible. When an **OPEN** statement that specifies the external unit identifiers previously indicated is executed followed by a **CLOSE** statement, the next input/output statements for external unit identifiers 0, 5, and 6 detect an error because the unit is not connected to files.

In the following example, **WRITE** statement (a) outputs data to the standard output file; **WRITE** statement (b) outputs data to the file named DATA6; and **WRITE** statement (c) outputs an error.

Example:

```
WRITE(6, *) A, B, C  ------(a) Standard output file
    ...
    ...
OPEN(6, FILE = "DATA6")
WRITE(6, *) I, J, K   ------(b) DATA6
    ...
CLOSE(6)
    ...
WRITE(6, *) X, Y, Z  ------(c) Unit 6 is not connected
```

9.4.3.2 Other File Preconnection

A file named fort.n is preconnected to each external unit identifier (n) other than 0, 5, and 6. Even if the **FILE** specifier is used in an **OPEN** statement, the executions of a **CLOSE** statement and an **OPEN** statement with the **FILE** specifier fort.n still allow unit *n* to be connected to fort.*n*.

In the following example, **WRITE** statement (a) outputs data to the file named fort.8; **WRITE** statement (b) outputs data to the file named DATA8; and **WRITE** statement (c) outputs data again to the file named fort.8. The records output by (a) are rewritten by (c).

See the description of the environment variable **VE_FORT***n* in “2.2 Environment Variables Referenced During Execution” to change a preconnection file.

Example:

```
WRITE(8, *) A, B, C  ------(a) fort.8
    ...
    ...
OPEN(8, FILE = "DATA8")
WRITE(8, *) I, J, K  ------(b) DATA8
    ...
CLOSE(8)
    ...
OPEN(8, FILE = "fort.8")
WRITE(8, *) X, Y, Z  ------(c) fort.8
```

9.4.4 Unnamed File

An unnamed file can be created by executing the **OPEN** statement with STATUS=”SCRATCH”. An unnamed file is created by the directory P_tmpdir in the header file <stdio.h>. However, if this directory cannot be accessed, the directory /tmp is used.

By using the environment variable **TMPDIR**, an unnamed file can be created in a specified directory.

9.4.5 Rounding Mode

The rounding mode can be specified by the **ROUND** specifier and the round edit specifier in an **OPEN** statement and a data transfer I/O statement. When these specifications are not set, the rounding mode is set to **PROCESSOR_DEFINED**. The value resulting from conversion in each mode is as follows.

ROUND specifier	edit descriptors	Conversion result
UP	RU	The smallest representable value that is greater than or equal to the original value
DOWN	RD	The largest representable value that is less than or equal to the original value
ZERO	RZ	The value closest to the original value and no greater in magnitude than the original value
NEAREST	RN	The closer of the two nearest representable values if one is closer than the other. When two values are equally close, it is rounded to the even one
COMPATIBLE	RC	The closer of the two nearest representable values or the value away from zero if halfway between them
PROCESSOR_DEFINED	RP	Same as **NEAREST**

9.4.6 NAMELIST Input Format

The NAMELIST input format supports the addition of "$" and "&" as the front character of the NAMELIST name. "$end", "&end" and "/" are supported as the end symbol.

9.4.7 NAMELIST Output Format

- Output of numeric-type array
  When two or more same values in a numeric array are consecutive, NAMELIST is output collectively form (Repeat* Value). This form can be changed by the environment variable **VE_FORT_NML_REPEAT_FORM**. See “2.2 Environment Variables Referenced During Execution” for details.

- **DELIM** specifier and character-type array
  When "NONE" is specified to **DELIM** specifier, characters are not separated from each other by value separators. When "QUOTE" or "APOSTROPHE" is specified to **DELIM** specifier, the same consecutive characters or strings are output collectively form (Repeat * Value). When **DELIM** specifier is omitted, characters are output continuously. This form can be changed by the environment variable

VE_FORT_NML_DELIM_BLANK. See “2.2 Environment Variables Referenced During Execution” for details.

Note NAMELIST output records produced with a **DELIM** specifier with a value of "NONE" and which contain a character sequence might not be acceptable as NAMELIST input records. If you want to use the output result of this text as input to the program, either specify other than "NONE" to **DELIM** specifier or set "YES" for environment variable **VE_FORT_NML_DELIM_BLANK** without **DELIM** specifier.

- Compatibility with compiler version 3.0.7

  If you want to NAMELIST output form of version 3.0.7 or earlier, set "NO" to environment variable **VE_FORT_NML_REPEAT_FORM**.

9.5 Fortran 2018 Extensions

This appendix describes the Fortran 2018 Extensions supported by NEC Fortran Compiler.

9.5.1 Execution Control

- The expression in an **ERROR STOP** or **STOP** statement can be used.
- The **ERROR STOP** and **STOP** statements have an optional **QUIET** specifier.

  Example:

```
STOP 13, QUIET = .True.
```

The above program exits normally with status of 13.

9.5.2 Intrinsic Procedures and Modules

- The intrinsic subroutine **MOVE_ALLOC** has optional STAT and ERRMSG arguments.

  Example:

```
INTEGER,ALLOCATABLE :: X(:),Y(:)
INTEGER ISTAT
CHARACTER(80) EMSG
...
CALL MOVE_ALLOC(X,Y,ISTAT,EMSG)
IF (ISTAT/=0) THEN
PRINT *,'UNEXPECTED ERROR IN MOVE_ALLOC: ',TRIM(EMSG)
```

- The argument DIM to the intrinsic procedures **ALL**, **ANY**, **FINDLOC**, **IALL**, **IANY**,

IPARITY, **MAXLOC**, **MAXVAL**, **MINLOC**, **MINVAL**, **NORM2**, **PARITY**, **PRODUCT** and **SUM** can be an optional dummy argument

Example:

```
SUBROUTINE SUB(X,N)
  REAL,INTENT(IN) :: X(:,:,:)
  INTEGER,INTENT(IN),OPTIONAL :: N
  IF (PRESENT(N)) THEN
    PRINT *,NORM2(X,N) ! RANK TWO ARRAY RESULT.
  ELSE
    PRINT *,NORM2(X)   ! SCALAR RESULT.
  END IF
END SUBROUTINE
```

- The intrinsic procedure **RANK** can be used. It returns the dimensionality of its argument.

  Example:

```
INTEGER I(3,3),RESULT
RESULT=RANK(I)
END
```

- The intrinsic procedure **REDUCE** can be used. It performs user-defined array reductions.

  Example:

```
MODULE TRIPLET_M
  TYPE TRIPLET
    INTEGER I,J,K
  END TYPE
CONTAINS
  PURE TYPE(TRIPLET) FUNCTION TADD(A,B)
    TYPE(TRIPLET),INTENT(IN) :: A,B
    TADD%I = A%I + B%I
    TADD%J = A%J + B%J
    TADD%K = A%K + B%K
  END FUNCTION
END MODULE
PROGRAM REDUCE_EXAMPLE
  USE TRIPLET_M
  TYPE(TRIPLET) A(2,3)
  A = RESHAPE( [ TRIPLET(1,2,3),TRIPLET(1,2,4), &
                 TRIPLET(2,2,5),TRIPLET(2,2,6), &
                 TRIPLET(3,2,7),TRIPLET(3,2,8) ], [ 2,3 ] )
  PRINT 1, REDUCE(A,TADD)
```

```
  PRINT 1, REDUCE(A,TADD,1)
  PRINT 1, REDUCE(A,TADD,A%I/=2)
  PRINT 1, REDUCE(ARRAY=A,DIM=2,OPERATION=TADD)
  PRINT 1, REDUCE(A, MASK=A%I/=2, DIM=1, OPERATION=TADD,
IDENTITY=TRIPLET(0,0,0))
1 FORMAT(1X,6('TRIPLET(',I0,',',I0,',',I0,')',:,'; '))
END PROGRAM
```

9.5.3 Input/Output

- The **RECL** specifier in an **INQUIRE** statement for an unconnected unit or file assigns the value -1 to the variable. For a unit or file connected with ACCESS="STREAM", it assigns the value −2 to the variable. Under previous Fortran standards, the variable became undefined.
- The SIZE= specifier can be used in a READ statement without ADVANCE='NO'.

  Example:

```
CHARACTER(65536) BUF
INTEGER NC
READ(*,'(A)',SIZE=NC) BUF
  PRINT *,'THE NUMBER OF CHARACTERS ON THAT LINE WAS',NC
```

9.5.4 Programs and Procedures

- If a dummy argument of a function that is part of an OPERATOR generic has the **VALUE** attribute, it is no longer required to have the **INTENT(IN)** attribute.

  Example:

```
MODULE MOD
  INTERFACE OPERATOR(+)
    MODULE PROCEDURE PLUS
  END INTERFACE
CONTAINS
  PURE INTEGER FUNCTION PLUS(A,B)
    INTEGER,VALUE :: A
    LOGICAL,VALUE :: B
    PLUS = MERGE(A+1,A,B)
  END FUNCTION
END MODULE
```

- If the second argument of a subroutine that is part of an ASSIGNMENT generic has the **VALUE** attribute, it is no longer required to have the **INTENT(IN)** attribute.

Example:

```
MODULE MOD
  INTERFACE ASSIGNMENT(=)
    MODULE PROCEDURE ASGN
  END INTERFACE
CONTAINS
  PURE SUBROUTINE ASGN(A,B)
    INTEGER,INTENT(OUT) :: A
    LOGICAL,VALUE :: B
    A = MERGE(1,0,B)
  END SUBROUTINE
END MODULE
```

9.5.5 Language-Mixed Programming

- A procedure argument of the C_FUNLOC function from the intrinsic module ISO_C_BINDING is no longer required to have the **BIND(C)** attribute.
- The **TYPE(*)** type specifier can be used. It must not have the ALLOCATABLE, CODIMENSION, INTENT (OUT), POINTER, or VALUE attribute.

  Example:

  Fortran program:

```
PROGRAM TYPE_STAR_EXAMPLE

  INTERFACE
    FUNCTION CHECKSUM(SCALAR,SIZE) BIND(C)
      USE ISO_C_BINDING
      TYPE(*) SCALAR
      INTEGER(C_INT),VALUE :: SIZE
      INTEGER(C_INT) CHECKSUM
    END FUNCTION
  END INTERFACE
  TYPE MYVEC3
    DOUBLE PRECISION V(3)
  END TYPE
  TYPE(MYVEC3) X
  CALL RANDOM_NUMBER(X%V)
  PRINT *,CHECKSUM(X,STORAGE_SIZE(X)/8)
END PROGRAM
```

C program:

```
int checksum(void *a,int n)
{
  int i;
  int res = 0;
  unsigned char *p = a;
  for (i=0; i<n; i++) res = 0x3fffffff&((res<<1) + p[i]);
  return res;
}
```

- A BIND(C) procedure can have optional arguments. The arguments cannot also have the VALUE attribute.

  Example:

  Fortran program:

```
PROGRAM OPTIONAL_EXAMPLE

  USE ISO_C_BINDING
  INTERFACE
    FUNCTION F(A,B) BIND(C)
      IMPORT
      INTEGER(C_INT),INTENT(IN) :: A
      INTEGER(C_INT),INTENT(IN),OPTIONAL :: B
      INTEGER(C_INT) F
    END FUNCTION
  END INTERFACE
  INTEGER(C_INT) X,Y
  X = F(3,14)
  Y = F(23)
  PRINT *,X,Y
END PROGRAM
```

C program:

```
int f(int *arg1,int *arg2)
{
  int res = *arg1;
  if (arg2) res += *arg2;
  return res;
}
```

9.5.6 Obsolescent features

- The **EQUIVALENCE**, **COMMON** and **BLOCK DATA** statement are considered to be obsolescent in Fortran 2018 standards, and will be reported as such if the **–std**=f2018 option is used.

9.6 Restrictions

- If the return value of a function has procedure pointer, the **RESULT** clause can not be used.
- Execution of SPMD (Single Program Multiple Data) programming model using coarray is limited to a single image. There is no parallel execution.

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.

C program (file name: a.c)

```
#include <stdlib.h>
#define N 1024
#define SIZE sizeof(double)
main()
{
  double *x = (double *)malloc(SIZE*N);
  double *y = (double *)malloc(SIZE*N);
  double *z = (double *)malloc(SIZE*N);
  int n;
  n = read_data(x, y);
  compute_(x, y, z, &n);
  write_data(z, n);
}
```

C program (file name: b.c)

```
#include <stdio.h>
int read_data(double *x, double *y)
{ ... }
```

Fortran program (file name: c.f90)

```
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
```

C program (file name: d.c)

```
int check_value_(double *x)
{ ... }
```

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** (*1)
	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** (*1)

Data Type	Fortran	C/C++
	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**

(*1) 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
! Define a derived type with the BIND attribute
TYPE, BIND(C) :: STR_TYPE
    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 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
...
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 See Section 8.3 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[21];              // Allocate 20 bytes + 1 byte for terminating
...
chfunc_(a, 20L);              // Call Fortran procedure
...
```

Callee (Fortran procedure):

```
CHARACTER*20 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
...
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)      // 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 arguments or the size (in bytes) of the return value for a procedure name arguments are added to the end of the arguments.

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 Library Reference

This chapter describes the original intrinsic procedures.

11.1 Intrinsic Procedures

11.1.1 ABS(*A*) Specific Name

FUNCTION

Returns the absolute value.

CLASS

Elemental function.

ARGUMENT

A:　*A* must be of Integer type, real type or complex type.

TYPE AND TYPE PARAMETER OF RESULT

When *A* is of complex type, the result is of real type with the same kind type parameter as *A*. Otherwise, the result is of the same type as *A*.

RESULT VALUE

When *A* is of integer or real type, the value of the result is |*A*| (absolute value of *A*). When *A* is the complex number (x,y), the value of the result is (x**2 + y**2)**(1/2).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BABS	INTEGER(1)	INTEGER(1)	
IIABS, HABS	INTEGER(2)	INTEGER(2)	
IABS	default integer	default integer	✓
JIABS	INTEGER(4)	INTEGER(4)	
KIABS	INTEGER(8)	INTEGER(8)	
ABS	default real	default real	✓
DABS	double precision real	double precision real	✓
QABS	REAL(16)	REAL(16)	
CABS	default complex	default real	✓
CDABS	double complex	double precision real	

Specific name	Argument Type	Result Type	Standard
ZABS	COMPLEX(8)	REAL(8)	
CQABS	COMPLEX(16)	REAL(16)	

11.1.2 ACOS(*X*) Specific Name

FUNCTION

Arccosine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type. Its value must satisfy |*X*| <= 1.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of arccos(X) expressed in radians.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ACOS	default real	default real	✓
DACOS	double precision real	double precision real	✓
QACOS, QARCOS	REAL(16)	REAL(16)	

11.1.3 ACOSH(*X*) Specific Name

FUNCTION

Hyperbolic arccosine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the hyperbolic arccosine, arccosh(*X*).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ACOSH	default real	default real	
DACOSH	double precision real	double precision real	
QACOSH	REAL(16)	REAL(16)	

11.1.4 AIMAG(*Z*) Specific Name

FUNCTION

Returns the imaginary part of a complex number.

CLASS

Elemental function.

ARGUMENT

Z: *A* must be of complex type.

TYPE AND TYPE PARAMETER OF RESULT

Real type with the same kind type parameter as *Z*.

RESULT VALUE

When the value of *A* is (*x*,*y*), the value of the result is *y*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
AIMAG	default complex	default real	
DIMAG	double complex	double precision real	
QIMAG	COMPLEX(16)	REAL(16)	

11.1.5 AINT(*A*) Specific Name

FUNCTION

Truncates to an integer value.

CLASS

Elemental function.

ARGUMENT

A: *A* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *A*.

RESULT VALUE

If $|A| < 1$, AINT (A) has the value 0.

If $|A| \geq 1$, AINT (A) has a value equal to the integer whose magnitude is the largest integer that does not exceed the magnitude of A and whose sign is the same as the sign of A.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
AINT	default real	default real	✓
DINT	double precision real	double precision real	✓
QINT	REAL(16)	REAL(16)	

11.1.6 AMT(*X*)

FUNCTION

Fetches the mantissa portion.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the mantissa of *X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
AMT	default real	default real	
DMT	double precision real	double precision real	
QMT	REAL(16)	REAL(16)	

11.1.7 AND(*I*,*J*)

This function is alias of IAND. See Section 11.1.44 for details.

11.1.8 ANINT(*A*) Specific Name

FUNCTION

Returns the nearest integer value (by rounding).

CLASS

Elemental function.

ARGUMENT

A: *A* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *A*.

RESULT VALUE

If *A* > 0, ANINT (A) has the value AINT(*A*+0.5).

If *A* <= 0, ANINT (A) has the value AINT(*A*-0.5).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ANINT	default real	default real	✓
DNINT	double precision real	double precision real	✓
QNINT	REAL(16)	REAL(16)	

11.1.9 ASIN(*X*) Specific Name

FUNCTION

Arcsine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type. Its value must satisfy |*X*| <= 1.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of arcsin(X) expressed in radians.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ASIN	default real	default real	✓

Specific name	Argument Type	Result Type	Standard
DASIN	double precision real	double precision real	✓
QASIN	REAL(16)	REAL(16)	

11.1.10 ASINH(*X*) Specific Name

FUNCTION

Hyperbolic arcsine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the hyperbolic arcsine, arcsinh(*X*).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ASINH	default real	default real	
DASINH	double precision real	double precision real	
QASINH	REAL(16)	REAL(16)	

11.1.11 ATAN(*X*) Specific Name

FUNCTION

Arctangent function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of arctan(X) expressed in radians.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ATAN	default real	default real	✓
DATAN	double precision real	double precision real	✓
QATAN	REAL(16)	REAL(16)	

11.1.12 ATAN2(*Y*,*X*) Specific Name

FUNCTION

Arctangent function.

CLASS

Elemental function.

ARGUMENT

Y: *Y* must be of real type.

X: *X* must be of the same type and kind type parameter as *Y*. If *Y* has the value zero, *X* shall not have the value zero.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The result has a value equal to the argument of the complex number (Y, X) expressed in radians.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ATAN2	REAL(4)	REAL(4)	✓
DATAN2	REAL(8)	REAL(8)	✓
QATAN2	REAL(16)	REAL(16)	

11.1.13 ATANH(*X*) Specific Name

FUNCTION

Hyperbolic arctangent function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the hyperbolic arctangent, arctanh(*X*).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ATANH	default real	default real	
DATANH	double precision real	double precision real	
QATANH	REAL(16)	REAL(16)	

11.1.14 BTEST(*I*,*POS*) Specific Name

FUNCTION

Tests a bit of an integer value.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of integer type.

POS: *POS* must be of integer type. Its value must be greater than or equal to zero and less than **BIT_SIZE**(*I*).

TYPE AND TYPE PARAMETER OF RESULT

Default logical type.

RESULT VALUE

If the *POS* bit of *I* is 1, the value of the result is true. If the *POS* bit of *I* is 0, the value of the result is false.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BBTEST	INTEGER(1)	INTEGER(1)	
BITEST, HTEST	INTEGER(2)	INTEGER(2)	
BTEST, BJTEST	INTEGER(4)	INTEGER(4)	
BKTEST	INTEGER(8)	INTEGER(8)	

11.1.15 CANG(*X*)

FUNCTION

Argument of a complex number.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of complex type.

TYPE AND TYPE PARAMETER OF RESULT

Real type with the same kind type parameter as *X*.

RESULT VALUE

The value of the result is the value of the argument of the complex number *X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
CANG	default complex	default real	
CDANG, ZANG	double complex	double precision real	

11.1.16 CBRT(*X*)

FUNCTION

Cube root.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the cube root of *X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
CBRT	default real	default real	
DCBRT	double precision real	double precision real	

Specific name	Argument Type	Result Type	Standard
QCBRT	REAL(16)	REAL(16)	

11.1.17 CLOCK(*D*)

FUNCTION

Obtains the CPU time.

CLASS

Subroutine.

ARGUMENT

D: *D* must be a scalar variable of double precision real or quadruple precision real type. It is an **INTENT(OUT)** argument. The accumulated CPU execution time (units in seconds, precision up to microseconds) from the time program execution begins until the subroutine referenced is set.

11.1.18 CONJG(*Z*) Specific Name

FUNCTION

Conjugates a complex number.

CLASS

Elemental function.

ARGUMENT

Z: *Z* must be of complex type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *Z*.

RESULT VALUE

If *Z* has the value (x, y), the result has the value (x,−y).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
CONJG	default complex	default complex	
DCONJG	double complex	double complex	
QCONJG	COMPLEX(16)	COMPLEX(16)	

11.1.19 COS(*X*) Specific Name

FUNCTION

Cosine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type or complex type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of cos(*X*). When *X* is of real type, the value is considered to be a value in radians. Note that when type parameter is single precision and absolute value of *X* is greater than $2^{21}\times\pi$, the value of the result is NaN. When *X* is of complex type, its real part is considered to be a value in radians. Note that when type parameter is single precision and absolute value of the argument is greater than $2^{21}\times\pi$, the value of the result is NaN.

See Section 11.5 for notes on other type parameters.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
COS	default real	default real	✓
DCOS	double precision real	double precision real	✓
QCOS	REAL(16)	REAL(16)	
CCOS	default complex	default complex	✓
CDCOS	COMPLEX(8)	COMPLEX(8)	
ZCOS	double complex	double complex	
CQCOS	COMPLEX(16)	COMPLEX(16)	

11.1.20 COSD(*X*)

FUNCTION

Cosine.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the cosine, cos(*X*), when *X* is a value in degrees. Note that when the absolute value of *X* is greater than $2^{21}\times180$, the value of the result is NaN.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
COSD	default real	default real	
DCOSD	double precision real	double precision real	
QCOSD	REAL(16)	REAL(16)	

11.1.21 COSH(*X*) Specific Name

FUNCTION

Hyperbolic cosine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of cosh(*X*), when *X* is a value in radians.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
COSH	default real	default real	✓
DCOSH	double precision real	double precision real	✓
QCOSH	REAL(16)	REAL(16)	

11.1.22 COTAN(*X*)

FUNCTION

Cotangent.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the cotangent, cotan(*X*). Note that when the absolute value of the argument is greater than $2^{50}\times 180$, the value of the result is NaN.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
COTAN	default real	default real	
DCOTAN	double precision real	double precision real	
QCOTAN	REAL(16)	REAL(16)	

11.1.23 DATE(*A*)

FUNCTION

Obtains the date.

CLASS

Subroutine.

ARGUMENT

A: *A* must be a scalar variable of default character type having a length of eight characters. It is an **INTENT(OUT)** argument. The value of the date is set in "*yy-mm-dd*" format.

11.1.24 DATIM(*A,B,C*)

FUNCTION

Obtains the date and time.

CLASS

Subroutine.

ARGUMENT

A: *A* must be a scalar variable of default character type having a length of eight characters. It is an **INTENT(OUT)** argument. The value of the date is set in the format specified by argument *C*.

B: *B* must be a scalar variable of default real type or of default character type having a length of eight characters. It is an **INTENT(OUT)** argument. If it is of default real type, the current time is set in hours. If it is of default character type, the current time is set in the format "*hh*:*mm*:*ss*".

C(optional): *C (optional)* must be a scalar of default integer type. It is an **INTENT(IN)** argument. It specifies the format of the date to be returned in argument *A*.

1	*yy-mm-dd* (default)
3	*mm/dd/yy*
4	*dd/mm/yy*

11.1.25 DBLE(*A*) Specific Name

FUNCTION

Converts to double precision real type.

CLASS

Elemental function.

ARGUMENT

A: *A* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Double precision real type.

RESULT VALUE

The result has the value REAL(A,KIND(0.0D0)).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
DBLE	default real	double precision real	✓
DBLEQ	REAL(16)	double precision real	

11.1.26 DCMPLX(*X*,*Y*)

FUNCTION

Converts to double precision complex type.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of integer type, real type, or complex type.

Y (optional): *Y* (optional) must be of integer type or real type. If *X* is of complex type, *Y* must not be specified.

TYPE AND TYPE PARAMETER OF RESULT

Double precision complex type.

RESULT VALUE

The value of the result is the value of CMPLX(*X*,*Y*,KIND=KIND(0.0D0)).

11.1.27 DFACT(*I*)

FUNCTION

Factorial.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of default integer type.

TYPE AND TYPE PARAMETER OF RESULT

Double precision real type.

RESULT VALUE

The value of the result is the value of *I* factorial converted to double precision real type.

11.1.28 DFLOAT(*A*)

FUNCTION

Converts to double precision real type.

CLASS

Elemental function.

ARGUMENT

A: *A* must be of integer type.

TYPE AND TYPE PARAMETER OF RESULT

Double precision real type.

RESULT VALUE

The value of the result is the value of REAL(*A*,KIND=KIND(0.0D0)).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
DFLOTI	INTEGER(2)	double precision real	
DFLOTJ	default integer	double precision real	
DFLOTK	INTEGER(8)	double precision real	

11.1.29 DIM(*X*,*Y*) Specific Name

FUNCTION

Returns the value *X*-*Y* if the difference of *X*-*Y* is positive, and otherwise returns zero.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of Integer type or real type.

Y: *Y* must be of the same type as *X* with the same kind type parameter as *X*.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is *X*-*Y* if *X* > *Y* and is zero if *X* <= *Y*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BDIM	INTEGER(1)	INTEGER(1)	
IIDIM, HDIM	INTEGER(2)	INTEGER(2)	
IDIM	default integer	default integer	✓
JIDIM	INTEGER(4)	INTEGER(4)	
KIDIM	INTEGER(8)	INTEGER(8)	
DIM	default real	default real	✓
DDIM	double precision real	double precision real	✓
QDIM	REAL(16)	REAL(16)	

11.1.30 DREAL(*A*)

FUNCTION

Converts to double precision real type.

CLASS

Elemental function.

ARGUMENT

A: *A* must be of complex type.

TYPE AND TYPE PARAMETER OF RESULT

Double precision real type.

RESULT VALUE

When the value of the *A* is (*x*,*y*), the value of the result is *x*.

11.1.31 ERF(*X*) Specific Name

FUNCTION

Error function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the error function of *X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ERF	default real	default real	
DERF	double precision real	double precision real	
QERF	REAL(16)	REAL(16)	

11.1.32 ERFC(*X*) Specific Name

FUNCTION

Complementary error function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value obtained when the value of the error function of *X* is subtracted from 1.0.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ERFC	default real	default real	
DERFC	double precision real	double precision real	
QERFC	REAL(16)	REAL(16)	

11.1.33 ETIME(*D*)

FUNCTION

Execution time.

CLASS

Subroutine.

ARGUMENT

D: *D* must be of double precision real-type. It is an **INTENT(OUT)** argument. The elapsed time (units in seconds) since System start.

NOTE

See Section 11.4.811.4.47 for details when used by function.

11.1.34 EXIT(*X*)

FUNCTION

Terminates execution of an executable program.

CLASS

Subroutine.

ARGUMENT

X: *X* must be a scalar of integer-type. It is an **INTENT(IN)** argument. The value *X* is returned as a program termination code.

11.1.35 EXP(*X*) Specific Name

FUNCTION

Exponential.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type or complex type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of e***X*. If X is of complex type, the value of the imaginary part is in radians. Note that when type parameter is single precision and absolute value of the argument is greater than $2^{21} \times \pi$, the value of the result is NaN.

See Section 11.5 for notes on other type parameters.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
EXP	default real	default real	✓
DEXP	double precision real	double precision real	✓
QEXP	REAL(16)	REAL(16)	
CEXP	default complex	default complex	✓
CDEXP	double complex	double complex	
ZEXP	COMPLEX(8)	COMPLEX(8)	
CQEXP	COMPLEX(16)	COMPLEX(16)	

11.1.36 EXP10(*X*)

FUNCTION

Exponential.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of 10.0***X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
EXP10	default real	default real	
DEXP10	double precision real	double precision real	
QEXP10	REAL(16)	REAL(16)	

11.1.37 EXP2(*X*)

FUNCTION

Exponential.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of 2.0***X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
EXP2	default real	default real	
DEXP2	double precision real	double precision real	
QEXP2	REAL(16)	REAL(16)	

11.1.38 EXPC(*X*)

FUNCTION

Exponential.

CLASS

Elemental function.

ARGUMENT

X:　*X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of e***X*-1.0.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
EXPC	default real	default real	
DEXPC	double precision real	double precision real	

11.1.39　EXPC10(*X*)

FUNCTION

Exponential.

CLASS

Elemental function.

ARGUMENT

X:　*X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of 10.0***X*-1.0.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
EXPC10	default real	default real	
DXPC10	double precision real	double precision real	
QXPC10	REAL(16)	REAL(16)	

11.1.40　EXPC2(*X*)

FUNCTION

Exponential.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X.*

RESULT VALUE

The value of the result is the value of 2.0***X*-1.0.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
EXPC2	default real	default real	
DEXPC2	double precision real	double precision real	
QEXPC2	REAL(16)	REAL(16)	

11.1.41 FACT(*I*)

FUNCTION

Factorial.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of default integer type.

TYPE AND TYPE PARAMETER OF RESULT

Default real type.

RESULT VALUE

The value of the result is the value of *I* factorial converted to default real type.

11.1.42 FLUSH(*UNIT*)

FUNCTION

Outputs the contents of the buffer.

CLASS

Subroutine.

ARGUMENT

UNIT: *UNIT* must be of integer type. It is an **INTENT(IN)** argument. *UNIT* is the external unit identifier to a file.

11.1.43 GAMMA(*X*) Specific Name

FUNCTION

Gamma function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the Gamma function of *X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
GAMMA	default real	default real	✓
DGAMMA	double precision real	double precision real	

11.1.44 IAND(*I*,*J*) Specific Name

FUNCTION

Bitwise logical AND.

CLASS

Elemental function.

ARGUMENT

I: I must be of Integer type.

J: J must be of integer type with the same kind type parameter as *I*.

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result is obtained by combining *I* and *J* bit-by-bit according to the following truth table:

I	*J*	IAND(*I*,*J*)
1	1	1
1	0	0

I	*J*	IAND(*I*,*J*)
0	1	0
0	0	0

NOTE

There may even be three or more arguments. In this case, the third and subsequent arguments must be of integer type with the same kind type parameter as *I*. Also, no keyword can be specified for the arguments.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BIAND	INTEGER(1)	INTEGER(1)	
IIAND, HIAND	INTEGER(2)	INTEGER(2)	
JIAND	INTEGER(4)	INTEGER(4)	
KIAND	INTEGER(8)	INTEGER(8)	

11.1.45 IBCLR(*I*,*POS*) Specific Name

FUNCTION

Sets one bit to zero.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of integer type.

POS: *POS* must be of integer type. Its value must be greater than or equal to zero and less than **BIT_SIZE**(*I*).

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result has the *POS* bit of I set to zero.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BBCLR	INTEGER(1)	INTEGER(1)	
IIBCLR, HBCLR	INTEGER(2)	INTEGER(2)	
JIBCLR	INTEGER(4)	INTEGER(4)	
KIBCLR	INTEGER(8)	INTEGER(8)	

11.1.46　IBITS(*I*,*POS*,*LEN*) Specific Name

FUNCTION

Extracts a sequence of bits.

CLASS

Elemental function.

ARGUMENT

I:　*I* must be of integer type.

POS:　*POS* must be of integer type. Its value must be nonnegative and *POS*+*LEN* must be less than or equal to **BIT_SIZE**(*I*).

LEN:　*LEN* must be of integer type. Its value must be nonnegative.

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result has *LEN* bits starting with the *POS* bit of I left justified with the remaining bits set to zero.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BBITS	INTEGER(1)	INTEGER(1)	
IIBITS, HBITS	INTEGER(2)	INTEGER(2)	
JIBITS	INTEGER(4)	INTEGER(4)	
KIBITS	INTEGER(8)	INTEGER(8)	

11.1.47　IBSET(*I*,*POS*) Specific Name

FUNCTION

Sets one bit to 1.

CLASS

Elemental function.

ARGUMENT

I:　*I* must be of integer type.

POS:　*POS* must be of integer type. Its value must be nonnegative and less than **BIT_SIZE**(*I*).

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result has the *POS* bit of *I* set to 1.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BBSET	INTEGER(1)	INTEGER(1)	
IIBSET, HBSET	INTEGER(2)	INTEGER(2)	
JIBSET	INTEGER(4)	INTEGER(4)	
KIBSET	INTEGER(8)	INTEGER(8)	

11.1.48 IEOR(*I*,*J*) Specific Name

FUNCTION

Bitwise logical OR.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of Integer type.

J: *J* must be of integer type with the same kind type parameter as *I*.

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result is obtained by combining *I* and *J* bit-by-bit according to the following truth table:

I	*J*	IEOR(*I*,*J*)
1	1	1
1	0	1
0	1	1
0	0	0

NOTE

There may even be three or more arguments. In this case, the third and subsequent arguments must be of integer type with the same kind type parameter as *I*. Also, no keyword can be specified for the arguments.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BIEOR, BIXOR	INTEGER(1)	INTEGER(1)	
IIEOR, HIEOR, HIXOR, IIXOR	INTEGER(2)	INTEGER(2)	
JIEOR, JIXOR	INTEGER(4)	INTEGER(4)	
KIEOR	INTEGER(8)	INTEGER(8)	

11.1.49 IMAG(*A*)

This function is alias of AIMAG. See Section 11.1.4 for details.

11.1.50 INT(*A*[,*KIND*]) Specific Name

FUNCTION

Converts to integer type (by truncating).

CLASS

Elemental function.

ARGUMENT

A: *A* must be of integer type, real type, or complex type.

KIND(optional): *KIND* must be a scalar integer initialization expression.

TYPE AND TYPE PARAMETER OF RESULT

Integer type. When *KIND* is specified, the kind type parameter is determined according to the KIND specification. When *KIND* is omitted, the kind type parameter is that of default integer type.

RESULT VALUE

If *A* is of integer type, the value of INT(*A*) is *A*.

If *A* is of real type and |*A*|<1, INT(*A*) is zero. If A is of real type and |A| >= 1, the value of INT(*A*) is the greatest integer less than or equal to the absolute value of *A* with the same sign of *A*.

If *A* is of complex type, the value of INT(*A*) is obtained by applying the rule described in Case 2 to the real part of *A*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
INT1	INTEGER(*), REAL(*), COMPLEX(*)	INTEGER(1)	

Specific name	Argument Type	Result Type	Standard
IJINT	INTEGER(4)	INTEGER(2)	
INT2	INTEGER(*), REAL(*), COMPLEX(*)	INTEGER(2)	
IIFIX, IINT, IINT, HFIX	REAL(4)	INTEGER(2)	
IIDINT	REAL(8)	INTEGER(2)	
IIQINT	REAL(16)	INTEGER(2)	
INT4, JFIX	INTEGER(*), REAL(*), COMPLEX(*)	default integer	
JIFIX	REAL(*)	default integer	
INT, JINT	default real	default integer	✓(INT only)
IDINT, JIDINT	double precision real	default integer	✓(IDINT only)
IQINT, JIQINT	REAL(16)	default integer	
INT8	INTEGER(*), REAL(*), COMPLEX(*)	INTEGER(8)	
KIFIX, KINT	REAL(4)	INTEGER(8)	
KIDINT	REAL(8)	INTEGER(8)	
KIQINT	REAL(16)	INTEGER(8)	

11.1.51 IOR(*I*,*J*) Specific Name

FUNCTION

Bitwise logical OR.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of Integer type.

J: *J* must be of integer type with the same kind type parameter as *I*.

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result is obtained by combining *I* and *J* bit-by-bit according to the following truth table:

I	*J*	IOR(*I*,*J*)
1	1	1
1	0	1
0	1	1
0	0	0

NOTE

There may even be three or more arguments. In this case, the third and subsequent arguments must be of integer type with the same kind type parameter as *I*. Also, no keyword can be specified for the arguments.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BIOR	INTEGER(1)	INTEGER(1)	
IIOR, HIOR	INTEGER(2)	INTEGER(2)	
JIOR	INTEGER(4)	INTEGER(4)	
KIOR	INTEGER(8)	INTEGER(8)	

11.1.52 IRE(*X*)

FUNCTION

Extracts the exponent part.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Default integer type.

RESULT VALUE

The value of the result is the exponent part of *X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
IRE	defauilt real	default integer	
IDE	double precision real	default integer	

Specific name	Argument Type	Result Type	Standard
IQE	REAL(16)	default integer	

11.1.53 ISHFT(*I*,*SHIFT*) Specific Name

FUNCTION

Logical shift.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of integer type.

SHIFT: *SHIFT* must be of integer type. Its absolute value must be less than or equal to **BIT_SIZE**(*I*).

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result is obtained by shifting the bits of *I* by *SHIFT* positions.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BSHFT	INTEGER(1)	INTEGER(1)	
IISHFT, HSHFT	INTEGER(2)	INTEGER(2)	
JISHFT	INTEGER(4)	INTEGER(4)	
KISHFT	INTEGER(8)	INTEGER(8)	

11.1.54 ISHFT(*I*,*SHIFT*[,*SIZE*]) Specific Name

FUNCTION

Performs a circular shift of the rightmost sequence of bits.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of integer type.

SHIFT: *SHIFT* must be of integer type. Its absolute value must be less than or equal to *SIZE*.

SIZE(optional): *SIZE* must be of integer type. The value of *SIZE* must be positive and must be less than or equal to **BIT_SIZE**(*I*). If *SIZE* is omitted, the value of

BIT_SIZE(I) is assumed to have been specified.

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result is obtained by circularly shifting the *SIZE* rightmost bits of *I* by *SHIFT* positions.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BSHFTC	INTEGER(1)	INTEGER(1)	
IISHFTC, HSHFTC	INTEGER(2)	INTEGER(2)	
JISHFTC	INTEGER(4)	INTEGER(4)	
KISHFTC	INTEGER(8)	INTEGER(8)	

11.1.55 ISNAN(*X*)

FUNCTION

Tests whether real numbers are NaN values.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Default logical type.

RESULT VALUE

If x is NaN, the result is .TRUE.; otherwise, the result is .FALSE..

11.1.56 IXOR(*I*,*J*)

This function is alias of IEOR. See Section 11.1.48 for details.

11.1.57 LGAMMA(*X*)

FUNCTION

Logarithmic Gamma function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the logarithmic Gamma function of *X*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ALGAMA	default real	default real	
DLGAMA	double precision real	double precision real	

11.1.58 LOC(*X*)

FUNCTION

Gets an address.

CLASS

Transformational function.

ARGUMENT

X: *X* must be a variable of any type.

TYPE AND TYPE PARAMETER OF RESULT

8byte integer type.

RESULT VALUE

The value of the result is the value of the address of *X*.

11.1.59 LOG(*X*) Specific Name

FUNCTION

Natural logarithm.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type or complex type. If *X* is of real type, its value must be positive. If *X* is of complex type, its value must not be (0.0,0.0).

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of loge(*X*). The value of a result of complex type is the principal value having an imaginary part w in the range -pi < w <= pi. The imaginary part of the result is pi only when the real part of the argument is negative and the imaginary part is 0.0.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ALOG	default real	default real	✓
DLOG	double precision real	double precision real	✓
QLOG	REAL(16)	REAL(16)	
CLOG	default complex	default complex	✓
CDLOG	double complex	double complex	
ZLOG	COMPLEX(8)	COMPLEX(8)	
CQLOG	COMPLEX(16)	COMPLEX(16)	

11.1.60 LOG10(*X*) Specific Name

FUNCTION

Common logarithm.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the logarithm log10(*X*).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ALOG10	default real	default real	✓
DLOG10	double precision real	double precision real	✓
QLOG10	REAL(16)	REAL(16)	

11.1.61 LOG2(*X*)

FUNCTION

Logarithm.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the logarithm log2(*X*).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ALOG2	default real	default real	
DLOG2	double precision real	double precision real	

11.1.62 MAX(*A1*,*A2*[,*A3*,…]) Specific Name

FUNCTION

Selects the maximum value.

CLASS

Elemental function.

ARGUMENT

A*n*: A*n* must all be of the same integer type or real type and must all have the same kind type parameter.

TYPE AND TYPE PARAMETER OF RESULT

Same as A*n*.

RESULT VALUE

The value of the result is the maximum argument value.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
IMAX0	INTEGER(2)	INTEGER(2)	
AIMAX0	INTEGER(2)	default real	

Specific name	Argument Type	Result Type	Standard
MAX0, JMAX0	default integer	default integer	✓(MAX0 only)
AMAX0, AJMAX0	default integer	default real	✓(AMAX0 only)
DMAX0	default integer	double precision real	
KMAX0	INTEGER(8)	INTEGER(8)	
AKMAX0	INTEGER(8)	default real	
IMAX1	default real	INTEGER(2)	
MAX1, JMAX1	default real	default integer	✓(MAX1 only)
KMAX1	default real	INTEGER(8)	
AMAX1	default real	default real	✓
DMAX1	double precision real	double precision real	✓

11.1.63 MAXVL()

FUNCTION

Obtains the maximum vector register length.

CLASS

Inquiry function.

TYPE AND TYPE PARAMETER OF RESULT

Default integer type.

RESULT VALUE

The value of the result is the maximum vector register length of the system.

11.1.64 MIN(*A1*,*A2*[,*A3*,…])

FUNCTION

Selects the minimum value.

CLASS

Elemental function.

ARGUMENT

A*n*: A*n* must all be of the same integer type or real type and must all have the same kind type parameter.

TYPE AND TYPE PARAMETER OF RESULT

Same as A*n*.

RESULT VALUE

The value of the result is the minimum argument value.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
IMIN0	INTEGER(2)	INTEGER(2)	
AIMIN0	INTEGER(2)	default real	
MIN0, JMIN0	default integer	default integer	✓(MAX0 only)
AMIN0, AJMIN0	default integer	default real	✓(AMAX0 only)
DMIN0	default integer	double precision real	
KMIN0	INTEGER(8)	INTEGER(8)	
AKMIN0	INTEGER(8)	default real	
IMIN1	default real	INTEGER(2)	
MIN1, JMIN1	default real	default integer	✓(MAX1 only)
KMIN1	default real	INTEGER(8)	
AMIN1	default real	default real	✓
DMIN1	REAL(8)	double precision real	✓

11.1.65 MOD(*A*,*P*) Specific Name

FUNCTION

Remainder function.

CLASS

Elemental function.

ARGUMENT

A: *A* must be of integer type or real type.

P: *P* must be of the same type and kind type parameter as *A*.

TYPE AND TYPE PARAMETER OF RESULT

Same as *A*.

RESULT VALUE

If *P* /= 0, the value of the result is A-INT(*A*/*P*)**P*. If *P* = 0, the result is undefined.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BMOD	INTEGER(1)	INTEGER(1)	

Specific name	Argument Type	Result Type	Standard
IMOD, HMOD	INTEGER(2)	INTEGER(2)	
MOD	default integer	default integer	✓
JMOD	INTEGER(4)	INTEGER(4)	
KMOD	INTEGER(8)	INTEGER(8)	
AMOD	default real	default real	✓
DMOD	double precision real	double precision real	✓
QMOD	REAL(16)	REAL(16)	

11.1.66 MVBITS(*FROM*,*FROMPOS*,*LEN*,*TO*,*TOPOS*) Specific Name

FUNCTION

Copies a bit sequence from one data object to another data object.

CLASS

Elemental subroutine.

ARGUMENT

FROM:　*FROM* must be of integer type. It is an **INTENT(IN)** argument.

FROMPOS:　*FROMPOS* must be of integer type and must be nonnegative. It is an **INTENT(IN)** argument. *FROMPOS+LEN* must be less than or equal to **BIT_SIZE**(*FROM*).

LEN:　*LEN* must be of integer type and must be nonnegative. It is an **INTENT(IN)** argument.

TO: *TO* must be of integer type with the same kind type parameter as *FROM* and may be the same variable as *FROM*. It is an **INTENT(INOUT)** argument. The bit string of length *LEN* starting at the position *FROMPOS* of *FROM* is copied to the position *TOPOS* of *TO*. No other bits of *TO* are changed. When control returns from the subroutine, the LEN bits of *TO* starting at *TOPOS* are equal to the value that the *LEN* bits of *FROM* starting at *FROMPOS* had when the subroutine was invoked.

TOPOS: *TOPOS* must be of integer type and must be nonnegative. It is an **INTENT(IN)** argument. *TOPOS+LEN* must be less than or equal to **BIT_SIZE**(*TO*).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BMVBITS	INTEGER(1)	-	
IMVBITS, HMVBITS	INTEGER(2)	-	
JMVBITS	INTEGER(4)	-	
KMVBITS	INTEGER(8)	-	

11.1.67 NINT(*A*[,*KIND*]) Specific Name

FUNCTION

Returns the nearest integer (by rounding).

CLASS

Elemental function.

ARGUMENT

A: *A* must be of real type.

KIND(optional): *KIND* must be a scalar integer initialization expression.

TYPE AND TYPE PARAMETER OF RESULT

Integer type. When *KIND* is specified, the kind type parameter is determined according to the KIND specification. When *KIND* is omitted, the kind type parameter is that of default integer type.

RESULT VALUE

When *A* > 0, the value of NINT(*A*) is INT(*A*+0.5). When *A* <= 0, the value of NINT(*A*) is INT(*A*-0.5).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
ININT	REAL(4)	INTEGER(2)	
NINT	default real	default integer	✓
JNINT	REAL(4)	INTEGER(4)	
KNINT	REAL(4)	INTEGER(8)	
IIDNNT	REAL(8)	INTEGER(2)	
IDNINT	double precision real	default integer	✓
JIDNNT	REAL(8)	INTEGER(4)	
KIDNNT	REAL(8)	INTEGER(8)	
IIQNNT	REAL(16)	INTEGER(2)	

Specific name	Argument Type	Result Type	Standard
IQNINT	REAL(16)	default integer	
JIQNNT	REAL(16)	INTEGER(4)	
KIQNNT	REAL(16)	INTEGER(8)	

11.1.68 NOT(*I*)

FUNCTION

Calculates the logical complement.

CLASS

Elemental function.

ARGUMENT

I: *I* must be of Integer type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *I*.

RESULT VALUE

The value of the result is obtained by taking the logical complement of *I* bit-by-bit according to the following truth table:

I	NOT(*I*)
1	0
0	1

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BNOT	INTEGER(1)	INTEGER(1)	
INOT, HNOT	INTEGER(2)	INTEGER(2)	
JNOT	INTEGER(4)	INTEGER(4)	
KNOT	INTEGER(8)	INTEGER(8)	

11.1.69 OR(*I*,*J*)

This function is alias of IOR. See Section 11.1.51 for details.

11.1.70 QCMPLX(*X*,*Y*)

FUNCTION

Converts to quadruple precision complex type.

CLASS

Elemental function.

ARGUMENT

X:　*X* must be of integer type, real type, or complex type.

Y (optional): *Y* (optional) must be of integer type or real type. If *X* is of complex type, *Y* must not be specified.

TYPE AND TYPE PARAMETER OF RESULT

Quadruple precision complex type.

RESULT VALUE

The value of the result is the value of CMPLX(*X*,*Y*,KIND=KIND(0.0Q0)).

11.1.71　QEXT(*X*)

FUNCTION

Converts to quadruple precision real type.

CLASS

Elemental function.

ARGUMENT

X:　*X* must be of integer type, real type, or complex type.

TYPE AND TYPE PARAMETER OF RESULT

Quadruple precision complex type.

RESULT VALUE

The value of the result is the value of REAL(*X*,KIND=KIND(0.0Q0)).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
QEXT	default real	REAL(16)	
QEXTD	REAL(8)	REAL(16)	

11.1.72　QFACT(*I*)

FUNCTION

Factorial.

CLASS

Elemental function.

ARGUMENT

I:　*I* must be of default integer type.

TYPE AND TYPE PARAMETER OF RESULT

Quadruple precision real type.

RESULT VALUE

The value of the result is the value of I factorial converted to quadruple precision real type.

11.1.73　QFLOAT(*A*)

FUNCTION

Converts to quadruple precision real type.

CLASS

Elemental function.

ARGUMENT

A:　*A* must be of integer type.

TYPE AND TYPE PARAMETER OF RESULT

Quadruple precision real type.

RESULT VALUE

The value of the result is the value of REAL(*A*,KIND=KIND(0.0Q0)).

11.1.74　QREAL(*A*)

FUNCTION

Converts to quadruple precision real type.

CLASS

Elemental function.

ARGUMENT

A:　*A* must be of quadruple complex type.

TYPE AND TYPE PARAMETER OF RESULT

Real type with the same kind type parameter as *A*.

RESULT VALUE

When the value of the *A* is (x,y), the value of the result is x.

11.1.75　REAL(*A*[,*KIND*])

FUNCTION

Converts to real type.

CLASS

Elemental function.

ARGUMENT

A: *A* must be of integer type, real type, or complex type.

KIND(optional): *KIND* must be a scalar integer initialization expression.

TYPE AND TYPE PARAMETER OF RESULT

Real type. When *A* is of integer type or real type and *KIND* is specified, the kind type parameter is determined according to the *KIND* specification. When *KIND* is omitted, the kind type parameter is the kind type parameter for default real type. When *A* is of complex type and *KIND* is specified, the kind type parameter is determined according to the KIND specification. When KIND is omitted, the kind type parameter is the kind type parameter of *A*.

RESULT VALUE

When *A* is of integer type or real type, the value of the result is the value of *A*. When *A* is of complex type, the value of the result is the value of the real part of *A*.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
FLOATI	INTEGER(2)	default real	
REAL, FLOAT	default integer	default real	✓
FLOATJ	INTEGER(4)	REAL(4)	
FLOATK	INTEGER(8)	default real	
SNGL	double precision	default real	✓
SNGLQ	REAL(16)	default real	

11.1.76 RSQRT(*X*)

FUNCTION

Reciprocal square root.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the approximate value of "1.0/sqrt(*X*)".

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
RSQRT	default real	default real	
DRSQRT	double precision real	double precision real	
QRSQRT	REAL(16)	REAL(16)	

11.1.77 SIGN(*A*,*B*) Specific Name

FUNCTION

The product of the absolute value of A and the sign of B.

CLASS

Elemental function.

ARGUMENT

A: *A* must be of integer type or real type.

B: *B* must be of the same type and kind type parameter as *A*.

TYPE AND TYPE PARAMETER OF RESULT

Same as *A*.

RESULT VALUE

The value of the result is ABS(*A*) when *B* >= 0, and it is -ABS(*A*) when *B* < 0.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
BSIGN	INTEGER(1)	INTEGER(1)	
IISIGN, HSIGN	INTEGER(2)	INTEGER(2)	
SIGN, ISIGN	default integer	default integer	✓
JISIGN	INTEGER(4)	INTEGER(4)	
KISIGN	INTEGER(8)	INTEGER(8)	
SIGN	default real	default real	✓
DSIGN	double precision real	double precision real	✓
QSIGN	REAL(16)	REAL(16)	

11.1.78 SIN(*X*) Specific Name

FUNCTION

Sine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type or complex type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of sin(*X*). When *X* is of real type, the value is considered to be a value in radians. Note that when type parameter is single precision and absolute value of *X* is greater than $2^{21}\times\pi$, the value of the result is NaN. When *X* is of complex type, its real part is considered to be a value in radians. Note that when type parameter is single precision and absolute value of the argument is greater than $2^{21}\times\pi$, the value of the result is NaN.

See Section 11.5 for notes on other type parameters.

RESULT VALUE

Specific name	Argument Type	Result Type	Standard
SIN	default real	default real	✓
DSIN	double precision real	double precision real	✓
QSIN	REAL(16)	REAL(16)	
CSIN	default complex	default complex	✓
CDSIN	double complex	double complex	
ZSIN	COMPLEX(8)	COMPLEX(8)	
CQSIN	COMPLEX(16)	COMPLEX(16)	

11.1.79 SIND(*X*)

FUNCTION

Sine.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of sin(X), when X is a value in degrees. Note that when the absolute value of X is greater than $2^{50}\times180$, the value of the result is NaN.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
SIND	default real	default real	
DSIND	REAL(8)	REAL(8)	
QSIND	REAL(16)	REAL(16)	

11.1.80 SINH(*X*) Specific Name

FUNCTION

Hyperbolic sine function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the hyperbolic sine, sinh(X).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
SINH	default real	default real	✓
DSINH	double precision real	double precision real	✓
QSINH	REAL(16)	REAL(16)	

11.1.81 SQRT(*X*) Specific Name

FUNCTION

Square root.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type or complex type. When *X* is of real type and not of complex type, the value must be greater than or equal to 0.0.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of sqrt(*X*). A result of complex type is the principal value with the real part greater than or equal to 0.0. If the real part of the result is 0.0, the imaginary part is greater than or equal to zero.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
SQRT	default real	default real	✓
DSQRT	double precision real	double precision real	✓
QSQRT	REAL(16)	REAL(16)	
CSQRT	default complex	default complex	✓
CDSQRT	double complex	double complex	
ZSQRT	COMPLEX(8)	COMPLEX(8)	
CQSQRT	COMPLEX(16)	COMPLEX(16)	

11.1.82 TAN(*X*) Specific Name

FUNCTION

Tangent function.

CLASS

Elemental function.

ARGUMENT

X: *X* must be of real type or complex type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of tan(*X*) expressed in radians. Note that when type parameter is single precision and absolute value of the argument is greater than $2^{21}\times\pi$, the value of the result is NaN.

See Section 11.5 for notes on other type parameters.

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
TAN	default real	default real	✓
DTAN	double precision real	double precision real	✓
QTAN	REAL(16)	REAL(16)	
CTAN	COMPLEX(4)	COMPLEX(4)	
CDTAN, ZTAN	COMPLEX(8)	COMPLEX(8)	
CQTAN	COMPLEX(16)	COMPLEX(16)	

11.1.83　TANH(*X*) Specific Name

FUNCTION

Hyperbolic tangent function.

CLASS

Elemental function.

ARGUMENT

X:　*X* must be of real type.

TYPE AND TYPE PARAMETER OF RESULT

Same as *X*.

RESULT VALUE

The value of the result is the value of the hyperbolic tangent, tanh(*X*).

SPECIFIC NAME

Specific name	Argument Type	Result Type	Standard
TANH	default real	default real	✓
DTANH	double precision real	double precision real	✓
QTANH	REAL(16)	REAL(16)	

11.1.84　TIME(*A*)

FUNCTION

Obtains the time.

CLASS

Subroutine.

ARGUMENT

A:　*A* must be a scalar variable of default character type with a length of eight characters. It is an **INTENT(OUT)** argument. It is set to the value of the time in the format "*hh*:*mm*:*ss*".

11.1.85　XOR(*I*,*J*)

This function is alias of IEOR. See Section 11.1.48 for details.

11.2 Matrix Multiply Library

Matrix multiply library is prepared for matrix-matrix or matrix-vector multiplication loops.

11.2.1 MATRIX-VECTOR Multiplication(*A*, *NAR*, *B*, *NBR*, *C*)

FUNCTION

MATRIX-VECTOR multiplication loops.

CLASS

Subroutine.

ARGUMENT

A:　*A* must be of integer type or real type two-dimensional array consisting.

NAR:　*NAR* must be of integer type.

B:　*B* must be of integer type or real type array consisting. This is same kind type parameter as *A*.

NBR:　*NBR* must be of integer type.

C:　*C* must be of integer type or real type array consisting. This is same kind type parameter as *A*. *C* is the result of MATRIX-VECTOR multiplication loops of *A* and *B*. Some functions are initialized with 0.

DETAIL

The combination of procedure name, initialize and each KIND is as follows.

Procedure (sum)	Procedure (difference)	KIND (*A*,*B*,*C*)	KIND (*NAR*,*NBR*)	Initialize (*C*)
VHMXV	VHSXV	REAL(KIND= 2)	INTEGER(KIND=4)	YES
VAMXV	VASXV	REAL(KIND= 4)	INTEGER(KIND=4)	YES
VDMXV	VDSXV	REAL(KIND= 8)	INTEGER(KIND=4)	YES
VQMXV	VQSXV	REAL(KIND=16)	INTEGER(KIND=4)	YES
VIMXV	VISXV	INTEGER(KIND=4)	INTEGER(KIND=4)	YES
VDMXVL	VDSXVL	REAL(KIND= 8)	INTEGER(KIND=8)	YES

Procedure (sum)	**Procedure (difference)**	**KIND (*A,B,C*)**	**KIND (*NAR,NBR*)**	**Initialize (*C*)**
VQMXVL	VQSXVL	REAL(KIND=16)	INTEGER(KIND=8)	YES
VLMXVL	VLSXVL	INTEGER(KIND=8)	INTEGER(KIND=8)	YES
VHMXP	VHSXP	REAL(KIND= 2)	INTEGER(KIND=4)	NO
VAMXP	VASXP	REAL(KIND= 4)	INTEGER(KIND=4)	NO
VDMXP	VDSXP	REAL(KIND= 8)	INTEGER(KIND=4)	NO
VQMXP	VQSXP	REAL(KIND=16)	INTEGER(KIND=4)	NO
VIMXP	VISXP	INTEGER(KIND=4)	INTEGER(KIND=4)	NO
VDMXPL	VDSXPL	REAL(KIND= 8)	INTEGER(KIND=8)	NO
VQMXPL	VQSXPL	REAL(KIND=16)	INTEGER(KIND=8)	NO
VLMXPL	VLSXPL	INTEGER(KIND=8)	INTEGER(KIND=8)	NO

The procedure with initialization "YES" is processed for sum and difference after the following processing.

```
DO I=1,NAR
  C(I)=0
ENDDO
```

The sum processing is as follows.

```
DO J=1,NBR
  DO I=1,NAR
    C(I) = C(I) + B(J) * A(I, J)
  ENDDO
ENDDO
```

The difference processing is as follows.

```
DO J=1,NBR
  DO I=1,NAR
    C(I) = C(I) - B(J) * A(I, J)
  ENDDO
ENDDO
```

11.2.2 MATRIX-VECTOR Multiplication(*A*, *NA*, *IAD*, *B*, *NB*, *C*, *NC*, *NAR*,

NBR)

FUNCTION

MATRIX-VECTOR multiplication loops.

CLASS

Subroutine.

ARGUMENT

A: *A* must be of integer type or real type two-dimensional array consisting.

NA: *NBR* must be of integer type. First stride.

IAD: *NBR* must be of integer type. Second stride.

B: *B* must be of integer type or real type array consisting. This is same kind type parameter as A.

NB: *NBR* must be of integer type. First stride.

C: *C* must be of integer type or real type array consisting. This is same kind type parameter as *A*. *C* is the result of MATRIX-VECTOR multiplication loops of *A* and *B*. Some functions are initialized with 0.

NC: *NBR* must be of integer type. First stride.

NAR: *NAR* must be of integer type.

NBR: *NBR* must be of integer type.

DETAIL

The combination of procedure name, initialize and each KIND is as follows.

Procedure (sum)	Procedure (difference)	KIND (*A*,*B*,*C*)	KIND (*NA*,*NB*,*NC*,*NAR*,*NBR*,*IAD*)	Initialize (*C*)
VHMXVA	VHSXVA	REAL(KIND= 2)	INTEGER(KIND=4)	YES
VAMXVA	VASXVA	REAL(KIND= 4)	INTEGER(KIND=4)	YES
VDMXVA	VDSXVA	REAL(KIND= 8)	INTEGER(KIND=4)	YES
VQMXVA	VQSXVA	REAL(KIND=16)	INTEGER(KIND=4)	YES
VIMXVA	VISXVA	INTEGER(KIND=4)	INTEGER(KIND=4)	YES
VDMVAL	VDSVAL	REAL(KIND= 8)	INTEGER(KIND=8)	YES
VQMVAL	VQSVAL	REAL(KIND=16)	INTEGER(KIND=8)	YES
VLMVAL	VLSVAL	INTEGER(KIND=8)	INTEGER(KIND=8)	YES
VHMXPA	VHSXPA	REAL(KIND= 2)	INTEGER(KIND=4)	NO
VAMXPA	VASXPA	REAL(KIND= 4)	INTEGER(KIND=4)	NO
VDMXPA	VDSXPA	REAL(KIND= 8)	INTEGER(KIND=4)	NO

Procedure (sum)	Procedure (difference)	KIND (*A,B,C*)	KIND (*NA,NB,NC,NAR,NBR,IAD*)	Initialize (*C*)
VQMXPA	VQSXPA	REAL(KIND=16)	INTEGER(KIND=4)	NO
VIMXPA	VISXPA	INTEGER(KIND=4)	INTEGER(KIND=4)	NO
VDMPAL	VDSPAL	REAL(KIND= 8)	INTEGER(KIND=8)	NO
VQMPAL	VQSPAL	REAL(KIND=16)	INTEGER(KIND=8)	NO
VLMPAL	VLSPAL	INTEGER(KIND=8)	INTEGER(KIND=8)	NO

The procedure with initialization "YES" is processed for sum and difference after the following processing.

```
DO I=1,NAR
  C(NC*I)=0
ENDDO
```

The sum processing is as follows.

```
DO J=1,NBR
  DO I=1,NAR
    C(NC*I) = C(NC*I) + B(NB*J) * A(NA*I, J)
  ENDDO
ENDDO
```

The difference processing is as follows.

```
DO J=1,NBR
  DO I=1,NAR
    C(NC*I) = C(NC*I) - B(NB*J) * A(NA*I, J)
  ENDDO
ENDDO
```

11.2.3 MATRIX- MATRIX Multiplication(*A*, *NA*, *IAD*, *B*, *NB*, *IBD*, *C*, *NC*, *ICD*, *NAR*, *NAC*, *NBC*)

FUNCTION

MATRIX- MATRIX multiplication loops.

CLASS

Subroutine.

ARGUMENT

A: *A* must be of integer type or real type two-dimensional array consisting.

NA: *NBR* must be of integer type. First stride.

IAD: *NBR* must be of integer type. Second stride.

B: *B* must be of integer type or real type array consisting. This is same kind type parameter as *A*.

NB: *NBR* must be of integer type. First stride.

IBD: *NBR* must be of integer type. Second stride.

C: *C* must be of integer type or real type array consisting. This is same kind type parameter as *A*. *C* is the result of MATRIX-VECTOR multiplication loops of *A* and *B*. Some functions are initialized with 0.

NC: *NBR* must be of integer type. First stride.

ICD: *NBR* must be of integer type. Second stride.

NAR: *NAR* must be of integer type.

NAC: *NBR* must be of integer type.

NBC: *NBR* must be of integer type.

DETAIL

The combination of procedure name, initialize and each KIND is as follows.

Procedure (sum)	**Procedure (difference)**	**KIND (*A,B,C*)**	**KIND (*NA,NB,NC,IAD,IBD,ICD, NAR,NAC,NBC*)**	**Initialize (*C*)**
VHMXMA	VHSXMA	REAL(KIND= 2)	INTEGER(KIND=4)	YES
VAMXMA	VASXMA	REAL(KIND= 4)	INTEGER(KIND=4)	YES
VDMXMA	VDSXMA	REAL(KIND= 8)	INTEGER(KIND=4)	YES
VQMXMA	VQSXMA	REAL(KIND=16)	INTEGER(KIND=4)	YES
VIMXMA	VISXMA	INTEGER(KIND=4)	INTEGER(KIND=4)	YES
VDMMAL	VDSMAL	REAL(KIND= 8)	INTEGER(KIND=8)	YES
VQMMAL	VQSMAL	REAL(KIND=16)	INTEGER(KIND=8)	YES
VLMMAL	VLSMAL	INTEGER(KIND=8)	INTEGER(KIND=8)	YES
VHMXQA	VHSXQA	REAL(KIND= 2)	INTEGER(KIND=4)	NO
VAMXQA	VASXQA	REAL(KIND= 4)	INTEGER(KIND=4)	NO
VDMXQA	VDSXQA	REAL(KIND= 8)	INTEGER(KIND=4)	NO
VQMXQA	VQSXQA	REAL(KIND=16)	INTEGER(KIND=4)	NO
VIMXQA	VISXQA	INTEGER(KIND=4)	INTEGER(KIND=4)	NO
VDMQAL	VDSQAL	REAL(KIND= 8)	INTEGER(KIND=8)	NO

Procedure (sum)	**Procedure (difference)**	**KIND (*A,B,C*)**	**KIND (*NA,NB,NC,IAD,IBD,ICD, NAR,NAC,NBC*)**	**Initialize (*C*)**
VQMQAL	VQSQAL	REAL(KIND=16)	INTEGER(KIND=8)	NO
VLMQAL	VLSQAL	INTEGER(KIND=8)	INTEGER(KIND=8)	NO

The procedure with initialization "YES" is processed for sum and difference after the following processing.

```
DO I=1,NAR
  C(NC*I)=0
ENDDO
```

The sum processing is as follows.

```
DO J=1,NBR
  DO I=1,NAR
    C(NC*I) = C(NC*I) + B(NB*J) * A(NA*I, J)
  ENDDO
ENDDO
```

The difference processing is as follows.

```
DO J=1,NBR
  DO I=1,NAR
    C(NC*I) = C(NC*I) - B(NB*J) * A(NA*I, J)
  ENDDO
ENDDO
```

11.3 UNIX System Function Interface

The UNIX-specific function can be used directly from Fortran program on UNIX system function interface. To use the UNIX system function interface, specify the modules described in following sections using **USE** statement or **-use** option.

Example:

USE statements:

```
PROGRAM MAIN
USE F90_UNIX
...
END PROGRAM MAIN
```

Compiler options:

```
$ nfort -use F90_UNIX,F90_UNIX_DIR a.f90
```

In the descriptions of the procedures, where it says KIND is (*), it means any kind of value.

When using each module with the **USE** statement or the **-use** compiler option, some variable names cannot be used. The variable names that cannot be used are as follows.

module	variable names
F90_UNIX	CLOCK_TICK_KIND, TMS
F90_UNIX_DIR	MODE_KIND
F90_UNIX_ENV	CLOCK_TICK_KIND, ID_KIND, LONG_KIND, SC_ARG_MAX, SC_CHILD_MAX, SC_CLK_TCK, SC_JOB_CONTROL, SC_NGROUPS_MAX, SC_OPEN_MAX, SC_SAVED_IDS, SC_STDERR_UNIT, SC_STDIN_UNIT, SC_STDOUT_UNIT, SC_STREAM_MAX, SC_TZNAME_MAX, SC_VERSION, TIME_KIND, TMS, UTSNAME
F90_UNIX_FILE	F_OK, ID_KIND, MODE_KIND, R_OK, STAT_T, S_IRGRP, S_IROTH, S_IRUSR, S_IRWXG, S_IRWXO, S_IRWXU, S_ISGID, S_ISUID, S_IWGRP, S_IWOTH, S_IWUSR, S_IXGRP, S_IXOTH, S_IXUSR, UTIMBUF, W_OK, X_OK
F90_UNIX_PROC	ATOMIC_INT, ATOMIC_LOG, PID_KIND, TIME_KIND, WNOHANG, WUNTRACED

When using each module with the **USE** statement or the **-use** compiler option, it uses other module of UNIX System Function Interface whole or necessary procedures. The modules and procedures used by each modules are as follows.

module	variable names
F90_UNIX	F90_UNIX_PROC : ABORT() F90_UNIX_ENV: GETPID(), GETUID(), GETGID(), IARGC(), HIDDEN_GETARG()=>GETARG(), CLOCK_TICK_KIND(), TIMES(), HIDDEN_GETENV()=>GETENV(), CLOCK_TICKS_PER_SECOND()=>CLK_TCK()
F90_UNIX_ENV	F90_UNIX_ERRNO (all procedures)

module	variable names
F90_UNIX_FILE	F90_UNIX_ENV (all procedures) F90_UNIX_ERRNO (all procedures)
F90_UNIX_PROC	F90_UNIX_ERRNO (all procedures)

"=>" indicate use a module procedure as another name.

11.3.1 F90_UNIX

The procedures provided by the F90_UNIX module are as follows.

SUBROUTINE ABORT(MESSAGE)

CHARACTER(*),OPTIONAL,INTENT(IN) :: MESSAGE

ABORT cleans up the I/O buffers and then terminates execution on UNIX systems. If *MESSAGE* is given it is written to logical unit 0 (zero) preceded by 'abort:'.

SUBROUTINE EXIT(STATUS)

INTEGER(*),OPTIONAL,INTENT(IN) :: STATUS

Terminate execution as if executing the **END** statement of the main program (or an unadorned STOP statement). If *STATUS* is given it is returned to the operating system (where applicable) as the execution status code. The integer kind can be used for argument *STATUS* only **INTEGER(KIND=4)** and **INTEGER(KIND=8)**.

SUBROUTINE FLUSH(LUNIT)

INTEGER(4),INTENT(IN) :: LUNIT

Flushes the output buffer of logical unit LUNIT. If *LUNIT* is not a valid unit number or is not connected to a file, error is raised.

SUBROUTINE FREE(IPTR)

INTEGER(8),INTENT(IN) :: IPTR

Frees the area specified with *IPTR*. *IPTR* must be the address of the area allocated with MALLOC.

SUBROUTINE GETARG(K,ARG)

INTEGER(4),INTENT(IN)::K

CHARACTER(*),INTENT(OUT)::ARG

See Section 11.3.3 for details of GETARG. When GETARG is used with this module, the option arguments LENARG and ERRNO cannot be used.

SUBROUTINE GETENV(NAME,VALUE)
CHARACTER(*),INTENT(IN)::NAME
CHARACTER(*),INTENT(OUT)::VALUE

See Section 11.3.3 for details of GETENV. When GETARG is used with this module, the option arguments LENVALUE and ERRNO cannot be used.

PURE INTEGER(4) FUNCTION GETGID()

Returns the group number of the calling process.

PURE INTEGER(4) FUNCTION GETPID()

Returns the process number of the calling process.

PURE INTEGER(4) FUNCTION GETUID()

Returns the user number of the calling process.

PURE INTEGER(4) FUNCTION IARGC()

Returns the number of command-line arguments; this is the same value as the intrinsic function COMMAND_ARGUMENT_COUNT, except that it returns -1 if even the program name is unavailable (the intrinsic function erroneously returns the same value, 0, whether the program name is available or not).

INTEGER(8) FUNCTION MALLOC(ISIZE)
INTEGER(*),INTENT(IN),VALUE :: ISIZE

Allocates necessary area size *ISIZE*. The starting address is returned (handled in units of bytes). This function is for byte pointer mode. The integer kind can be used for argument *ISIZE* only INTEGER(KIND=4) and INTEGER(KIND=8).

11.3.2 F90_UNIX_DIR

The procedures provided by the F90_UNIX_DIR module are as follows.

SUBROUTINE CHDIR(PATH,ERRNO)
CHARACTER(*),INTENT(IN) :: PATH
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Sets the current working directory to *PATH*. Note that any trailing blanks in *PATH* may be significant. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE GETCWD(PATH,LENPATH,ERRNO)
CHARACTER(*),OPTIONAL,INTENT(OUT) :: PATH
INTEGER(4),OPTIONAL,INTENT(OUT) :: LENPATH
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Accesses the current working directory information. If *PATH* is present, it receives the name of the current working directory, blank-padded or truncated as appropriate if the length of the current working directory name differs from that of PATH. If *LENPATH* is present, it receives the length of the current working directory name. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE LINK(EXISTING,NEW,ERRNO)
CHARACTER(*),INTENT(IN) :: EXISTING,NEW
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Creates a new link (with name given by NEW) for an existing file (named by *EXISTING*). If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE RENAME(OLD,NEW,ERRNO)
CHARACTER(*),INTENT(IN) :: OLD

CHARACTER(*),INTENT(IN) :: NEW

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Changes the name of the file *OLD* to *NEW*. Any existing file *NEW* is first removed. Note that any trailing blanks in *OLD* or *NEW* may be significant. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE UNLINK(PATH,ERRNO)

CHARACTER(*),INTENT(IN) :: PATH

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Deletes the file *PATH*. Note that any trailing blanks in *PATH* may be significant. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

11.3.3 F90_UNIX_ENV

The procedures provided by the F90_UNIX_ENV module are as follows.

SUBROUTINE GETARG(K,ARG,LENARG,ERRNO)

INTEGER(*),INTENT(IN) :: K

CHARACTER(*),OPTIONAL,INTENT(OUT) :: ARG

INTEGER(4),OPTIONAL,INTENT(OUT) :: LENARG

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Accesses command-line argument number *K*, where argument zero is the program name. If *ARG* is present, it receives the argument text (blank-padded or truncated as appropriate if the length of the argument differs from that of *ARG*). If *LENARG* is present, it receives the length of the argument. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE GETENV(NAME,VALUE,LENVALUE,ERRNO)

CHARACTER(*),INTENT(IN) :: NAME

CHARACTER(*),OPTIONAL,INTENT(OUT) :: VALUE

INTEGER(4),OPTIONAL,INTENT(OUT) :: LENVALUE

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Accesses the environment variable named by *NAME*. If *VALUE* is present, it receives the text value of the variable (blank-padded or truncated as appropriate if the length of the value differs from that of *VALUE*). If *LENVALUE* is present, it receives the length of the value. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

PURE SUBROUTINE GETHOSTNAME(NAME,LENNAME)

CHARACTER(*),OPTIONAL,INTENT(OUT) :: NAME

INTEGER(4),OPTIONAL,INTENT(OUT) :: LENNAME

If *NAME* is present it receives the text of the standard host name for the current processor, blank-padded or truncated if appropriate. If *LENNAME* is present it receives the length of the host name. If no host name is available *LENNAME* will be zero.

PURE SUBROUTINE GETLOGIN(S,LENS)

CHARACTER(*),OPTIONAL,INTENT(OUT) :: S

INTEGER(4),OPTIONAL,INTENT(OUT) :: LENS

Accesses the user name (login name) associated with the calling process. If *S* is present, it receives the text of the name (blank-padded or truncated as appropriate if the length of the login name differs from that of *S*). If *LENS* is present, it receives the length of the login name.

SUBROUTINE ISATTY(LUNIT,ANSWER,ERRNO)

INTEGER(*),INTENT(IN) :: LUNIT

LOGICAL(*),INTENT(OUT) :: ANSWER

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

ANSWER receives the value .TRUE. if and only if the logical unit identified by *LUNIT* is connected to a terminal. If *LUNIT* is not a valid unit number or is not connected to any file, error is raised. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE TIME(ITIME,ERRNO)
INTEGER(4),INTENT(OUT) :: ITIME
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

ITIME receives the operating system date/time in seconds since the Epoch. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE TTYNAME(LUNIT,S,LENS,ERRNO)
INTEGER(*),INTENT(IN) :: LUNIT
CHARACTER(*),OPTIONAL,INTENT(OUT) :: S
INTEGER(4),OPTIONAL,INTENT(OUT) :: LENS
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Accesses the name of the terminal connected to the logical unit identified by *LUNIT*. If *S* is present, it receives the text of the terminal name (blank-padded or truncated as appropriate, if the length of the terminal name differs from that of *S*). If *LENS* is present, it receives the length of the terminal name. If *LUNIT* is not a valid logical unit number, or is not connected, error is raised. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

11.3.4 F90_UNIX_ERRNO

The parameters provided by the F90_UNIX_ERRNO module are as follows.

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Many procedures provided by the UNIX system function interface have an optional *ERRNO* argument. If this argument is provided it receives the error status from the procedure; zero indicates successful completion, otherwise it will be a non-zero error code. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message. If a procedure has no *ERRNO* argument it indicates that procedure always succeeds.

11.3.5 F90_UNIX_FILE

The parameters provided by the F90_UNIX_FILE module are as follows.

INTEGER(4),PARAMETER :: F_OK

Flag for requesting file existence check.

INTEGER(4),PARAMETER :: R_OK

Flag for requesting file readability check.

INTEGER(4),PARAMETER :: S_IRGRP

File mode bit indicating group read permission.

INTEGER(4),PARAMETER :: S_IROTH

File mode bit indicating other read permission.

INTEGER(4),PARAMETER :: S_IRUSR

File mode bit indicating user read permission.

INTEGER(4),PARAMETER :: S_IRWXG

Mask to select the group accessibility bits from a file mode.

INTEGER(4),PARAMETER :: S_IRWXO

Mask to select the other accessibility bits from a file mode.

INTEGER(4),PARAMETER :: S_IRWXU

Mask to select the user accessibility bits from a file mode.

INTEGER(4),PARAMETER :: S_ISGID

File mode bit indicating that the file is set-group-ID.

INTEGER(4),PARAMETER :: S_ISUID

File mode bit indicating that the file is set-user-ID.

INTEGER(4),PARAMETER :: S_IWGRP

File mode bit indicating group write permission.

INTEGER(4),PARAMETER :: S_IWOTH

File mode bit indicating other write permission.

INTEGER(4),PARAMETER :: S_IWUSR

File mode bit indicating user write permission.

INTEGER(4),PARAMETER :: S_IXGRP

File mode bit indicating group execute permission.

INTEGER(4),PARAMETER :: S_IXOTH

File mode bit indicating other execute permission.

INTEGER(4),PARAMETER :: S_IXUSR

File mode bit indicating user execute permission.

INTEGER(4),PARAMETER :: W_OK

Flag for requesting file writability check.

INTEGER(4),PARAMETER :: X_OK

Flag for requesting file executability check.

The types provided by the F90_UNIX_FILE module are as follows.

STAT_T

```
TYPE STAT_T
    INTEGER(4) ST_MODE
    INTEGER(4) ST_INO
    INTEGER(4) ST_DEV
    INTEGER(4) ST_NLINK
    INTEGER(4) ST_UID
    INTEGER(4) ST_GID
    INTEGER(4) ST_SIZE
    INTEGER(4) ST_ATIME, ST_MTIME, ST_CTIME
END TYPE
```

Derived type holding file characteristics.

ST_MODE

File mode (read/write/execute permission for user/group/other, plus set-group-ID and set-user-ID bits).

ST_INO

File serial number.

ST_DEV

ID for the device on which the file resides.

ST_NLINK

The number of links to the file.

ST_UID

User number of the file's owner.

ST_GID

Group number of the file.

ST_SIZE

File size in bytes (regular files only).

ST_ATIME

Time of last access.

ST_MTIME

Time of last modification.

ST_CTIME

Time of last file status change.

The procedures provided by the F90_UNIX_FILE module are as follows.

PURE SUBROUTINE ACCESS(PATH,AMODE,ERRNO)

CHARACTER(*),INTENT(IN) :: PATH

INTEGER(*),INTENT(IN) :: AMODE

INTEGER(4),INTENT(OUT) :: ERRNO

Checks file accessibility according to the value of *AMODE*; this should be F_OK or a combination of R_OK, W_OK and X_OK. In the latter case the values may be combined by addition or the intrinsic function IOR.

The result of the accessibility check is returned in *ERRNO*, which receives zero for success or an error code indicating the reason for access rejection.

SUBROUTINE CHMOD(PATH,MODE,ERRNO)

CHARACTER(*),INTENT(IN) :: PATH

INTEGER(*),INTENT(IN) :: MODE

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Sets the file mode (ST_MODE) to *MODE*. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE FSTAT(LUNIT,BUF,ERRNO)
INTEGER(*),INTENT(IN) :: LUNIT
TYPE(STAT_T),INTENT(OUT) :: BUF
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

BUF receives the characteristics of the file connected to logical unit *LUNIT*. If *LUNIT* is not a valid logical unit number or is not connected to a file, error is raised. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE LSTAT(PATH,BUF,ERRNO)
CHARACTER(*),INTENT(IN) :: PATH
TYPE(STAT_T),INTENT(OUT) :: BUF
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

BUF receives the characteristics of the file *PATH*. If Path is link file, *BUF* receives the characteristics of the link. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE STAT(PATH,BUF,ERRNO)
CHARACTER(*),INTENT(IN) :: PATH
TYPE(STAT_T),INTENT(OUT) :: BUF
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

BUF receives the characteristics of the file *PATH*. If Path is link file, *BUF* receives the characteristics of the linked file. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

11.3.6 F90_UNIX_PROC

The procedures provided by the F90_UNIX_PROC module are as follows.

SUBROUTINE ALARM(SECONDS,SUBROUTINE,SECLEFT,ERRNO)
INTEGER(*),INTENT(IN) :: SECONDS
INTERFACE
SUBROUTINE SUBROUTINE()
END
END INTERFACE
OPTIONAL SUBROUTINE
INTEGER(4),OPTIONAL,INTENT(OUT) :: SECLEFT
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Establishes an “alarm” call to the procedure SUBROUTINE to occur after *SECONDS* seconds have passed, or cancels an existing alarm if *SECONDS*==0. If SUBROUTINE is not present, any previous association of a subroutine with the alarm signal is left unchanged. If *SECLEFT* is present, it receives the number of seconds that were left on the preceding alarm or zero if there were no existing alarm. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE EXECL(PATH,ARG0...,ERRNO)
CHARACTER(*),INTENT(IN) :: PATH
CHARACTER(*),INTENT(IN) :: ARG0...
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Executes a program (*PATH*) instead of the current image. The arguments to the new program are specified by the dummy arguments which are named *ARG0*, *ARG1*, etc. up to *ARG20*. Note that these are not optional arguments, any actual argument that is itself an optional dummy argument must be present. This function is the same as EXECV except that the arguments are provided individually instead of via an array; and because they are provided individually, there is no need to provide the lengths (the lengths being taken from each argument itself). If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE EXECLP(FILE,ARG0...,ERRNO)

CHARACTER(*),INTENT(IN) :: FILE

CHARACTER(*),INTENT(IN) :: ARG0...

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Executes a program (FILE) instead of the current image. The arguments to the new program are specified by the dummy arguments which are named *ARG0*, *ARG1*, etc. up to *ARG20*. Note that these are not optional arguments, any actual argument that is itself an optional dummy argument must be present. This function is the same as EXECL except that determination of the program to be executed follows the same rules as EXECVP. If ERRNO argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE EXECV(PATH,ARGV,LENARGV,ERRNO)

CHARACTER(*),INTENT(IN) :: PATH

CHARACTER(*),INTENT(IN) :: ARGV(:)

INTEGER(*),INTENT(IN) :: LENARGV(:)

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Executes the program (*PATH*) in place of the current process image. *ARGV* is the array of argument strings, *LENARGV* containing the desired length of each argument. If *ARGV* is not zero-sized, *ARGV(1)*(:*LENARGV(1)*) is passed as argument zero (i.e. the program name). If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE EXECVE(PATH,ARGV,LENARGV,ENV,LENENV,ERRNO)

CHARACTER(*),INTENT(IN) :: PATH

CHARACTER(*),INTENT(IN) :: ARGV(:)

INTEGER(*),INTENT(IN) :: LENARGV(:)

CHARACTER(*),INTENT(IN) :: ENV(:)

INTEGER(*),INTENT(IN) :: LENENV(:)

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Similar to **EXECV**, with the environment strings specified by *ENV* and *LENENV* being passed to the new program. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE EXECVP(FILE,ARGV,LENARGV,ERRNO)

CHARACTER(*),INTENT(IN) :: FILE

CHARACTER(*),INTENT(IN) :: ARGV(:)

INTEGER(*),INTENT(IN) :: LENARGV(:)

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

The same as **EXECV** except that the program to be executed, *FILE*, is searched for using the **PATH** environment variable (unless it contains a slash character, in which case **EXECVP** is identical in effect to **EXECV**). If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE FORK(PID,ERRNO)

INTEGER(4),INTENT(OUT) :: PID

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

Creates a new process which is an exact copy of the calling process. In the new process, the value returned in *PID* is zero; in the calling process the value returned in *PID* is the process ID of the new (child) process. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

PURE SUBROUTINE SLEEP(SECONDS,SECLEFT)

INTEGER(*),INTENT(IN) :: SECONDS

INTEGER(4),OPTIONAL,INTENT(OUT) :: SECLEFT

Suspends process execution for *SECONDS* seconds, or until a signal has been delivered. If *SECLEFT* is present, it receives the number of seconds remaining in the sleep time (zero unless the sleep was interrupted by a signal).

SUBROUTINE SYSTEM(STRING,STATUS,ERRNO)

CHARACTER(*),INTENT(IN) :: STRING

INTEGER(4)OPTIONAL,INTENT(OUT) :: STATUS,ERRNO

Passes *STRING* to the command processor for execution. If *STATUS* is present it receives the completion status. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

SUBROUTINE WAIT(STATUS,RETPID,ERRNO)

INTEGER(4),OPTIONAL,INTENT(OUT) :: STATUS

INTEGER(4),OPTIONAL,INTENT(OUT) :: RETPID

INTEGER(4,OPTIONAL,INTENT(OUT) :: ERRNO

Wait for any child process to terminate (returns immediately if one has already terminated).

If *STATUS* is present it receives the termination status of the child process. If *RETPID* is present it receives the process number of the child process. If *ERRNO* argument is provided, 0 is returned for normal termination. A non-zero error code is returned for abnormal termination. If the *ERRNO* argument is omitted and an error condition is raised, the program will be terminated with an informative error message.

11.4 Other Library

System functions that can be used in a C library can also be called from Fortran in these routines.

Fortran libraries are not intrinsic functions. Therefore, the compiler treats these libraries according to the **IMPLICIT** statement specification or the implicit type declarations (initial letters i, j, k, l, m, and n indicate integer type; other letters indicate real type). If the implicit type and the library's function type do not match, the type declaration for the function (e.g., CTIME) must be specified.

11.4.1 ABORT()

FUNCTION

Terminates a program abnormally.

CLASS

Subroutine.

11.4.2 ACCESS(*PATH,MODE*)

FUNCTION

Check user's permissions for a file.

CLASS

Function.

ARGUMENT

PATH: *PATH* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *PATH* is the file path to check.

MODE: *MODE* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *MODE* is the accessibility check pattern.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.3 ALARM(*SECS,PROC*)

FUNCTION

Sets an alarm clock of the process.

CLASS

Function.

ARGUMENT

SECS: *SECS* must be of 4-byte integer type. It is an INTENT(IN) argument. *SECS* is the alarm clock time (handled in units of seconds) of the process.

PROC:　*PROC* must be of External procedure name.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

The remaining seconds are returned when the function is called.

11.4.4 CHDIR(*PATH*)

FUNCTION

Changes the work directory.

CLASS

Function.

ARGUMENT

PATH:　*PATH* must be a scalar variable of default character type. It is an INTENT(IN) argument. *PATH* is the directory path to change.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.5 CHMOD(*NAME,MODE*)

FUNCTION

Changes the access mode.

CLASS

Function.

ARGUMENT

NAME:　*NAME* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *NAME* is the path to change access mode.

MODE:　*MODE* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *Mode* is the access mode to change.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.6 CTIME(*I*)

FUNCTION

Transform date and time to string.

CLASS

Function.

ARGUMENT

I: *I* must be of 4-byte integer type. It is an **INTENT(IN)** argument.

TYPE AND TYPE PARAMETER OF RESULT

Default Character type of length 24.

RESULT VALUE

Interprets *I* as a time since the Epoch, converts it to local time, and returns it in the following format:

Sun Jan. 19 01:03:52 1992

11.4.7 DTIME(*TARRAY*)

FUNCTION

Execution time.

CLASS

Function.

ARGUMENT

TARRAY: *TARRAY* must be of 4-byte real-type array consisting of two elements. It is an **INTENT(OUT)** argument. User time from the previous reference of this function is assigned to the first element of *TARRAY*. Sys time is assigned to the second element.

TYPE AND TYPE PARAMETER OF RESULT

4-byte real type.

RESULT VALUE

The value of the result is the sum of User time and Sys time.

11.4.8 ETIME(*TARRAY*)

FUNCTION

Execution time.

CLASS

Function.

ARGUMENT

TARRAY: *TARRAY* must be of 4-byte real-type array consisting of two elements. It is an **INTENT(OUT)** argument. User time from the beginning of the program is assigned to the first element of *TARRAY*. Sys time is assigned to the second element.

TYPE AND TYPE PARAMETER OF RESULT

4-byte real type.

RESULT VALUE

The value of the result is the sum of User time and Sys time (units in seconds).

NOTE

See Section 11.1.3311.4.47 for details when used by subroutine.

11.4.9 FDATE()

FUNCTION

Get the current time as a string.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

Default Character type of length 24.

RESULT VALUE

Returns current time in following format:

Sun Jan. 19 01:03:52 1992

NOTE

Also usable as a subroutine in the following format:

call FDATE (A)

A is Default Character type of length 24 and an **INTENT(OUT)** argument.

A is set current time in following format:

Sun Jan. 19 01:03:52 1992

11.4.10 FORK()

FUNCTION

Creates a new process.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

Process ID is returned for normal termination. Error code is returned for abnormal termination.

11.4.11 FREE(*ADDR*)

FUNCTION

Deallocate memory.

CLASS

Subroutine.

ARGUMENT

ADDR: *ADDR* must be of double precision integer type. It is an **INTENT(IN)** argument. *ADDR* is the address of the area allocated with MALLOC.

11.4.12 FREE2(*ADDR*)

FUNCTION

Deallocate memory.

CLASS

Subroutine.

ARGUMENT

ADDR: *ADDR* must be of double precision integer type. It is an **INTENT(IN)** argument. *ADDR* is the address of the area allocated with MALLOC2.

11.4.13 FSEEK(*UNIT,OFFSET,WHENCE*)

FUNCTION

Repositions a file.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument. *UNIT* is the external unit identifier to a file.

OFFSET: *OFFSET* must be of 4-byte integer-type. It is an **INTENT(IN)** argument. Offset in bytes, relative to *WHENCE*, that is to be the new location of the file marker.

WHENCE: *WHENCE* must be of 4-byte integer-type. It is an **INTENT(IN)** argument. A position in the file. It must be one of the following:

Value	Position
0	Positions the file relative to the beginning of the file.
1	Positions the file relative to the current position.
2	Positions the file relative to the end of the file.

TYPE AND TYPE PARAMETER OF RESULT

4-byte integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

NOTE

Also usable as a subroutine in the following format:

call FSEEK (UNIT,OFFSET,WHENCE)

11.4.14 FSTAT(*UNIT,SXBUF*)

FUNCTION

Get file status.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument.

UNIT is the external unit identifier to a file.

SXBUF: *SXBUF* must be of 4-byte integer-type array consisting of nineteen elements. It is an **INTENT(OUT)** argument. The status of the file is set in *SXBUF*.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

NOTE

The information of SXBUF is below.

SXBUF(1) Device the file resides on

SXBUF(2) File inode number

SXBUF(3) Access mode of the file

SXBUF(4) Number of hard links to the file

SXBUF(5) User ID of owner

SXBUF(6) Group ID of owner

SXBUF(7) 0

SXBUF(8) Size of the file (bytes)

SXBUF(9) Last access time

SXBUF(10) Last modification time

SXBUF(11) Last file status change time

SXBUF(12)-(19) Future Reserved

11.4.15 FTELL(*UNIT*)

FUNCTION

Return the current position of a file.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument.

UNIT is the external unit identifier to a file.

TYPE AND TYPE PARAMETER OF RESULT

4-byte integer type.

RESULT VALUE

The result is the offset, in bytes, from the beginning of the file. A negative value indicates an error.

11.4.16 FTELLI8(*UNIT*)

FUNCTION

Return the current position of a file.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument.

UNIT is the external unit identifier to a file.

TYPE AND TYPE PARAMETER OF RESULT

8-byte integer type.

RESULT VALUE

The result is the offset, in bytes, from the beginning of the file. A negative value indicates an error.

11.4.17 GETARG(*POS,VAL*)

FUNCTION

Get command line argument.

CLASS

Subroutine.

ARGUMENT

POS: *POS* must be of 4-byte integer-type. It is an **INTENT(IN)** argument. *POS* is the argument position.

VAL: *VAL* must be a scalar variable of default character type. It is an **INTENT(OUT)** argument. The string in the command line passed to the program is set in *VAL*.

11.4.18 GETCWD(*PATH*)

FUNCTION

Get current working directory.

CLASS

Function.

ARGUMENT

PATH: *PATH* must be a scalar variable of default character type. It is an **INTENT(OUT)** argument. The path of current working directory is set in *PATH*.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.19 GETENV(*NAME,VAL*)

FUNCTION

Get an environment variable.

CLASS

Subroutine.

ARGUMENT

NAME: *NAME* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *NAME* is the string of environment variable name.

VAL: *VAL* must be a scalar variable of default character type. It is an **INTENT(OUT)** argument. The value of environment variable is set in *VAL*.

NOTE

Also usable as a function in the following format.

Result type is integer type. The function returns a 1 if a match is found, and 0 otherwise.

```
INTEGER RESULT,GETENV
RESULT = GETENV(NAME,VAL)
```

11.4.20 GETGID()

FUNCTION

Get group id.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

Group ID is returned.

11.4.21 GETLOG(*NAME*)

FUNCTION

Get command line argument.

CLASS

Subroutine.

ARGUMENT

NAME: *NAME* must be a scalar variable of default character type. It is an **INTENT(OUT)** argument. The string of login user name is set in *NAME*.

11.4.22 GETPID()

FUNCTION

Get process id.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

Process ID is returned.

11.4.23 GETPOS(*UNIT*)

FUNCTION

Return the current position of a file.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument.

UNIT is the external unit identifier to a file.

TYPE AND TYPE PARAMETER OF RESULT

4-byte integer type.

RESULT VALUE

The result is the offset, in bytes, from the beginning of the file. A negative value indicates an error.

11.4.24 GETPOSI8(*UNIT*)

FUNCTION

Return the current position of a file.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument.

UNIT is the external unit identifier to a file.

TYPE AND TYPE PARAMETER OF RESULT

8-byte integer type.

RESULT VALUE

The result is the offset, in bytes, from the beginning of the file. A negative value indicates an error.

11.4.25 GETUID()

FUNCTION

Get user id.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

User ID is returned.

11.4.26 GMTIME(*I,IA9*)

FUNCTION

Transform date and time to 4-byte Integer-type array.

CLASS

Subroutine.

ARGUMENT

I: *I* must be of 4-byte integer type. It is an **INTENT(IN)** argument.

IA9: *IA9* must be of 4-byte integer-type array consisting of nine elements. It is an **INTENT(OUT)** argument. Interprets *I* as a time since the Epoch and numerical values of it are assigned to each element of *IA9*.

11.4.27 HOSTNM(*NAME*)

FUNCTION

Get hostname.

CLASS

Function.

ARGUMENT

NAME: *NAME* must be a scalar variable of default character type. It is an **INTENT(OUT)** argument. The host name is set in *NAME*.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.28 IARGC()

FUNCTION

Get command-line arguments.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

Number of arguments on the command line is returned.

11.4.29 IDATE(*IA3*)

FUNCTION

Transform date to 4-byte Integer-type array.

CLASS

Subroutine.

ARGUMENT

IA3: *IA3* must be of 4-byte integer-type array consisting of three elements. It is an **INTENT(OUT)** argument. Month, date, and year are assigned to each element of *IA3*, in this order.

11.4.30 IERRNO()

FUNCTION

Get the latest error code.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

4-byte integer type.

RESULT VALUE

Returns the number of the last detected error codes.

11.4.31 ISATTY(*UNIT*)

FUNCTION

Test whether unit connect to terminal equipment.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument.

UNIT is the external unit identifier.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

If it is connected to the terminal equipment, 1 is returned; otherwise, 0 is returned.

11.4.32 ITIME(*IA3*)

FUNCTION

Transform time to 4-byte Integer-type array.

CLASS

Subroutine.

ARGUMENT

IA3: *IA3* must be of 4-byte integer-type array consisting of three elements. It is an **INTENT(OUT)** argument. Hour, minute, and second are assigned to each element of *IA3*, in this order.

11.4.33 KILL(*PID,SIGNUM*)

FUNCTION

Send a signal to a process or process group.

CLASS

Function.

ARGUMENT

PID: *PID* must be of 4-byte integer type. It is an **INTENT(IN)** argument. Sends the signal to the process ID specified by argument *PID*.

SIGNUM: *SIGNUM* must be of 4-byte integer type. It is an **INTENT(IN)** argument. Sends the signal number specified by argument *SIGNUM*.

TYPE AND TYPE PARAMETER OF RESULT

4-byte integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.34 LINK(*PATH1,PATH2*)

FUNCTION

Create Link.

CLASS

Function.

ARGUMENT

PATH1: *PATH1* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *PATH1* is the path of an existing file.

PATH2: *PATH2* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *PATH2* is the path to be linked to the file.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.35 LSTAT(*PATH,SXBUF*)

FUNCTION

Get file status.

CLASS

Function.

ARGUMENT

PATH: *PATH* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *PATH* is the file path.

SXBUF: *SXBUF* must be of 4-byte integer-type array consisting of nineteen elements. It is an **INTENT(OUT)** argument. The status of the file is set in *SXBUF*. If *PATH* is link file, *SXBUF* receives the characteristics of the link.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

NOTE

The information of SXBUF is below.

SXBUF(1) Device the file resides on

SXBUF(2) File inode number

SXBUF(3) Access mode of the file

SXBUF(4) Number of hard links to the file

SXBUF(5) User ID of owner

SXBUF(6) Group ID of owner

SXBUF(7) 0

SXBUF(8) Size of the file (bytes)

SXBUF(9) Last access time

SXBUF(10) Last modification time

SXBUF(11) Last file status change time

SXBUF(12)-(19) Future Reserved

11.4.36 LTIME(*I,IA9*)

FUNCTION

Transform local date and time to 4-byte Integer-type array.

CLASS

Subroutine.

ARGUMENT

I: *I* must be of 4-byte integer type. It is an **INTENT(IN)** argument.

IA9: IA9 must be of 4-byte integer-type array consisting of nine elements. It is an **INTENT(OUT)** argument. Interprets I as a time since the Epoch. The time is converted to the local time, and numerical values of it are assigned to each element of *IA9*.

11.4.37 MALLOC(*SIZE*)

FUNCTION

Allocate memory.

CLASS

Function.

ARGUMENT

SIZE: *SIZE* must be of 4-byte integer type. It is an **INTENT(IN)** argument. *SIZE* is necessary area size (handled in units of bytes) to allocate.

TYPE AND TYPE PARAMETER OF RESULT

Double precision Integer type.

RESULT VALUE

Starting address of the memory allocated is returned.

11.4.38 MALLOC2(*SIZE*)

FUNCTION

Allocate memory.

CLASS

Function.

ARGUMENT

SIZE: *SIZE* must be of double precision integer type. It is an **INTENT(IN)** argument. *SIZE* is necessary area size (handled in units of bytes) to allocate.

TYPE AND TYPE PARAMETER OF RESULT

Double precision Integer type.

RESULT VALUE

Starting address of the memory allocated is returned.

11.4.39 PERROR(*A*)

FUNCTION

Print the latest error message to standard error output.

CLASS

Subroutine.

ARGUMENT

A: *A* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. The string of *A*, colon, margin, and error message are concatenated and printed to standard error output.

11.4.40 RENAME(*FROM,TO*)

FUNCTION

Rename a file.

CLASS

Function.

ARGUMENT

FROM: *FROM* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *FROM* is the path name of an existing file.

TO: *TO* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *TO* is the new path for this file.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.41 SECNDS(*T*)

FUNCTION

Get the elapsed time from reference time in seconds.

CLASS

Function.

ARGUMENT

T: *T* must be of 4-byte real type. It is an **INTENT(IN)** argument. *T* is a reference time, also in seconds.

TYPE AND TYPE PARAMETER OF RESULT

4-byte real type.

RESULT VALUE

The value of the result is elapsed time from argument T in seconds. If *T* is zero, time from midnight is returned.

11.4.42 SIGNAL(*SIGNUM,HANDLER*)

FUNCTION

Specifies the operation during signal reception.

CLASS

Function.

ARGUMENT

SIGNUM: *SIGNUM* must be of real type. It is an **INTENT(IN)** argument. Specify the signal number by argument *SIGNUM*.

HANDLER: *HANDLER* must be of External procedure name. It is an **INTENT(IN)** argument. Name of user signal handling function specified by *HANDLER*.

TYPE AND TYPE PARAMETER OF RESULT

4-byte integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.43 SLEEP(*SECS*)

FUNCTION

Suspend execution.

CLASS

Subroutine.

ARGUMENT

SECS: *SECS* must be of 4-byte integer type. It is an **INTENT(IN)** argument.

SECS is the time (handled in units of seconds) to suspend.

11.4.44 STAT(*UNIT,SXBUF*)

FUNCTION

Get file status.

CLASS

Function.

ARGUMENT

PATH: *PATH* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *PATH* is the file path.

SXBUF: *SXBUF* must be of 4-byte integer-type array consisting of nineteen elements. It is an **INTENT(OUT)** argument. The status of the file is set in *SXBUF*.

If *PATH* is link file, *SXBUF* receives the characteristics of the linked file.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

NOTE

The information of SXBUF is below.

SXBUF(1)	Device the file resides on
SXBUF(2)	File inode number
SXBUF(3)	Access mode of the file
SXBUF(4)	Number of hard links to the file
SXBUF(5)	User ID of owner
SXBUF(6)	Group ID of owner
SXBUF(7)	0

SXBUF(8) Size of the file (bytes)

SXBUF(9) Last access time

SXBUF(10) Last modification time

SXBUF(11) Last file status change time

SXBUF(12)-(19) Future Reserved

11.4.45 SYMLNK(*PATH1,PATH2*)

FUNCTION

Create a symbolic link.

CLASS

Function.

ARGUMENT

PATH1: *PATH1* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. Name of the path to be used by symbolic link *PATH2*.

PATH2: *PATH2* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. Name of a file(symbolic link name) to be created.

TYPE AND TYPE PARAMETER OF RESULT

4-byte integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.46 SYSTEM(*CMD*)

FUNCTION

Passes string to the command processor for execution.

CLASS

Function.

ARGUMENT

CMD: *CMD* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *CMD* is the string to the command processor for execution.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

Exit status of the *CMD* executed is returned.

NOTE

Also usable as a subroutine in the following format.

```
CALL SYSTEM(CMD)
```

11.4.47 TIME()

FUNCTION

Get time in seconds.

CLASS

Function.

TYPE AND TYPE PARAMETER OF RESULT

4-byte real type.

RESULT VALUE

Returns the value of time in seconds since the Epoch.

11.4.48 TTYNAM(*UNIT*)

FUNCTION

Get name of the terminal equipment.

CLASS

Function.

ARGUMENT

UNIT: *UNIT* must be of 4-byte integer-type. It is an **INTENT(IN)** argument.

UNIT is the external unit identifier.

TYPE AND TYPE PARAMETER OF RESULT

Default character type.

RESULT VALUE

Name of the terminal equipment connected to external unit identifier *UNIT* is returned.

11.4.49 UNLINK(*PATH*)

FUNCTION

Remove file.

CLASS

Function.

ARGUMENT

PATH: *PATH* must be a scalar variable of default character type. It is an **INTENT(IN)** argument. *PATH1* is the file path.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

0 is returned for normal termination. A non-zero error code is returned for abnormal termination.

11.4.50 WAIT(*STATUS*)

FUNCTION

Waits for a child process to stop or terminate.

CLASS

Function.

ARGUMENT

STATUS: *STATUS* must be a scalar variable of default character type. It is an **INTENT(OUT)** argument. The status of the child process is set in *STATUS*.

TYPE AND TYPE PARAMETER OF RESULT

Integer type.

RESULT VALUE

Child process ID is returned for normal termination. Error code is returned as a negative number for abnormal termination.

11.5 Notes

- Trigonometric and exponential functions fail to calculate results and in an error state when the value of an argument is within a certain range.
  The functions and their corresponding range of argument to be an error conditions is as follows.

  Single precision:

Function	Effective range
sin(x), cos(x), tan(x), cotan(x)	$\|x\| \geqq 2^{21} \times \pi$
sind(x), cosd(x)	$\|x\| \geqq 2^{21} \times 180$
csin(x+yi), ccos(x+yi)	$\|x\| \geqq 2^{21} \times \pi$
cexp(x+yi)	$\|x\| \geqq 2^{21} \times \pi$

Double precision:

Function	Effective range
dsin(x), dcos(x), dtan(x), dcotan(x)	$\lvert x\rvert \geqq 2^{50} \times \pi$
dsind(x), dcosd(x)	$\lvert x\rvert \geqq 2^{50} \times 180$
cdsin(x+yi), cdcos(x+yi)	$\lvert x\rvert \geqq 2^{50} \times \pi$
cdexp(x+yi)	$\lvert y\rvert \geqq 2^{50} \times \pi$

Quadruple precision:

Function	Effective range
qsin(x), qcos(x), qtan(x), qcotan(x)	$\lvert x\rvert \geqq 2^{110} \times \pi$
cqsin(x+yi), cqcos(x+yi)	$\lvert x\rvert \geqq 2^{110} \times \pi$
cqexp(x+yi)	$\lvert y\rvert \geqq 2^{110} \times \pi$

Chapter12 Messages

12.1 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.

12.1.1 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 outputting a module procedure name, the module name and procedure name are separated by "::".
- When outputting an internal procedure name, the host procedure name and the internal procedure name are separated by "::".
- When outputting a derived type component name, the variable name and

component name are separated by "%".

- 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.

12.1.2 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 t han the threshold value for vectorizing. The threshold value can be change d 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., .EQV., .NEQV., or .NOT. operation appears in the loop control expression.
- An equation operation (.EQ.. .NE., ==, /=) 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(1)” is required but its type is unvectorizable and the compiler cannot prepare any work vector.

```
subroutine sub(a,b,c,d,n,k)
complex(16) a(n)
complex(8) b(n),c(0:n),d(n)
do i=1,n
  a(1) = b(i) + d(i) + c(i)
  c(i) = a(1)
enddo
end
```

vec(109): Vectorization obstructive statement.

A loop cannot be vectorized because a statement that makes a whole loop unvectorizable appears.

vec(110): Vectorization obstructive procedure reference : Procedure-name

A loop cannot be vectorized because a procedure 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 procedure reference. : Procedure-name

A loop cannot be vectorized because there is a procedure 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 procedure calls.

A loop cannot be vectorized because there are too many procedure 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 derived-type assignment.

vec(163): Exception handling obstructs vectorization.

A loop cannot be vectorized because there are some expressions relaed to C++ exception handling.

vec(180): I/O statement obstructs vectorization.

A loop cannot be vectorized because there is an I/O statement.

vec(181): Allocation obstructs vectorization.

A loop cannot be vectorized because there is a memory allocation.

vec(182): Deallocation obstructs vectorization.

A loop cannot be vectorized because there is a memory deallocation.

vec(183): Run-time checking obstructs vectorization.

A loop cannot be vectorized because there is an expression to check run-time error or run-time status check. The expression is generated for not only **–fcheck** but also to check memory allocation, pointer status etc.

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(1017): Subroutine call prevents optimization.

Subroutine call prevents optimization.

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 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(1118): This I/O statement inhibits to optimization of loop.

An I/O statement inhibits optimization.

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(1300): Assumed-size private arrays inhibit concurrency.

Concurrency optimization is inhibited because of assumed-size array reference.

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.

opt(1382): Subroutine calls are handled only when "cncall" is used.

Concurrency optimization is inhibited because of user subroutine call. It may be optimized if **cncall** directive is specified.

opt(1387): Overlapping EQUIVALENCEd variables inhibit concurrency.

Optimization is inhibited because of overlapping equivalenced variables.

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.

12.2 Runtime Error Messages

12.2.1 Format

"Runtime Error:" is followed by line number, file name, and error message. Line number and file name may not be displayed.

Runtime Error:[*Line Number*, *File Name*:] *Error Message*

12.2.2 List of Error Messages

ADVANCE= specifier must be 'YES' or 'NO'

The value of the **ADVANCE** specifier in the **READ** statement or **WRITE** statement is incorrect. The **ADVANCE** specifier must be either 'YES' or 'NO'.

ALLOCATABLE *dimname* is not currently allocated

The allocatable array *dimname* is not currently allocated. The array *dimname* must be allocated.

ALLOCATE failed: Out of memory

Failed allocate due to out of memory. Check the memory size you are using and review the program.

Array constructor implied DO limit expression value *value* is out of range for index variable *var* type *type*

Array constructor implied DO limit expression value *value* is out of range for index variable *var* type *type*. Change the *value* of the DO limit expression.

Array constructor implied DO step expression value *value* is out of range for index variable *var* type *type*

Array constructor implied DO step expression value *value* is out of range for index variable *var* type *type*. Change the *value* of the DO step expression.

ASYNCHRONOUS= specifier must be 'NO' or 'YES'

The value of the **ASYNCHRONOUS** specifier in the **OPEN** statement is incorrect. The **ASYNCHRONOUS** specifier must be either 'YES' or 'NO'.

Buffer overflow on output

The record buffer overflowed in the I/O statement. Verify that the value specified in the environment variable **VE_FORT_FMTBUF**, **VE_FORT_RECORDBUF** or the **RECL** specifier in the OPEN statement are greater than the size of the output data.

Call to OMP_SET_MAX_ACTIVE_LEVELS from within a name region

The **OMP_SET_MAX_ACTIVE_LEVELS** was called from the name region. Check the program.

Cannot allocate ALLOCATABLE variable - out of memory

Failed reserve for temporary area for **ALLOCATABLE** variable due to out of memory. Check the memory size you are using and review the program.

Cannot allocate array temporary - out of memory

Failed reserve for temporary area for array due to out of memory. Check the memory size you are using and review the program.

Cannot allocate I/O buffer in OPEN processing UNIT=*unit-number*

The OPEN statement for this *unit-number* failed to reserve an I/O buffer. Close the unnecessary external unit identifier by **CLOSE** statement or changes the size of an I/O buffer at the environment variable **VE_FORT_SETBUF**.

Cannot allocate initial memory - out of memory

Failed reserve for temporary area for initial memory due to out of memory. Check the memory size you are using and review the program.

Cannot allocate memory for asynchronous i/o

Failed reserve for temporary area for asynchronous i/o. Change the number of the input/output item for input/output statement.

Cannot allocate memory for environment variable VE_FMTIO_OFFLOAD

Failed reserve for temporary area for environment variable **VE_FMTIO_OFFLOAD**. You must not specify environment variable **VE_FMT_OFFLOAD**.

Cannot allocate memory for environment variable VE_FORT_UFMTADJUST

Failed reserve for temporary area for environment variable **VE_FORT_UFMTADJUST**. Change array size of the input/output item for input/output statement.

Cannot allocate memory for environment variable VE_FORT_UFMTENDIAN

Failed reserve for temporary area for environment variable **VE_FORT_UFMTENDIAN**. Change array size of the input/output item for input/output statement.

Cannot allocate record buffer in OPEN processing UNIT=*unit-number*

The OPEN statement for this *unit-number* failed to reserve a record buffer. Close the unnecessary external unit identifier by **CLOSE** statement or changes the size of a record buffer at the environment variable **VE_FORT_RECORDBUF**.

Cannot BACKSPACE unformatted ACCESS='STREAM' unit *Unit Number*

BACKSPACE statements cannot be executed on an unformatting stream file. If you want to use an unformatting stream file, delete the **BACKSPACE** statement. If you want to execute the **BACKSPACE** statement, opens it to an unformatted sequential file.

Cannot find OLD file

The file name specified in the **OPEN** statement with **STATUS='OLD'** does not exist. Check the file name and correct if it is incorrect. If file name correct, correct the value of the **STATUS** specifier in the **OPEN** statement.

Cannot get storage for automatic array - out of memory

Failed reserve for temporary area for automatic array due to out of memory. Check the memory size you are using and review the program.

Cannot get storage for variable - out of memory

Failed reserve for temporary area for variable due to out of memory. Check the memory size you are using and review the program.

Character string edit descriptor does not terminate before format end

The character string edit descriptor is incorrect in the following manner. Correct the format specification.

- For H edit descriptor, there is no n characters following a character H.
- For character string edit descriptor, there is no a right delimiter.

Character string edit descriptor used on input

Do not specify the character string edit descriptor in a format specification of input statement. Correct the format specification of the input statement.

DECIMAL= specifier must be 'POINT' or 'COMMA'

The value of the **DECIMAL** specifier in the **OPEN**, **READ** or **WRITE** statement is incorrect. The **DECIMAL** specifier must be either 'POINT' or 'COMMA'.

DELIM= specifier in OPEN for an UNFORMATTED file

UNFORMATTED is specified for the **FORM** specifier in the **OPEN** statement. In this case, do not specify the **DELIM** specifier. If the external file is an unformatted file, delete the **DELIM** specifier. Otherwise, correct the value of the **FORM** specifier.

DIM argument (*value*) out of range 1:*rank* in intrinsic CSHIFT

The *value* of the DIM argument of intrinsic CSHIFT is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic EOSHIFT

The *value* of the DIM argument of intrinsic EOSHIFT is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic FINDLOC

The *value* of the DIM argument of intrinsic FINDLOC is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic LBOUND

The *value* of the DIM argument of intrinsic LBOUND is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic MAXLOC

The *value* of the DIM argument of intrinsic MAXLOC is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic MAXVAL

The *value* of the DIM argument of intrinsic MAXVAL is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic MINLOC

The *value* of the DIM argument of intrinsic MINLOC is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic MINVAL

The *value* of the DIM argument of intrinsic MINVAL is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic SIZE

The *value* of the DIM argument of intrinsic SIZE is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank* in intrinsic UBOUND

The *value* of the DIM argument of intrinsic UBOUND is out of range. Change the *value* of the DIM argument.

DIM argument (*value*) out of range 1:*rank*+1 in intrinsic SPREAD

The *value* of the DIM argument of intrinsic SPREAD is out of range. Change the *value* of the DIM argument.

Direct access is incompatible with the POSITION= specifier

DIRECT is specified for the **ACCESS** specifier in the **OPEN** statement. In this case, do not specify the **POSITION** specifier. If the external file is a direct file, delete the **POSITION** specifier. Otherwise, correct the value of the **ACCESS** specifier.

DO limit expression value *value* is out of range for index variable *var* type *type*

DO limit expression value *value* is out of range for index variable *var* type *type*. Change the *value* of the DO limit expression.

DO step expression value *value* is out of range for index variable *var* type *type*

DO step expression value *value* is out of range for index variable *var* type *type*. Change the *value* of the DO step expression.

Element *element* of ORDER argument (value *value*) to intrinsic RESHAPE is out of range (1:*rank*)

The *value* of the ORDER argument of intrinsic RESHAPE is out of range. Change the *value* of the ORDER argument.

ENDFILE applied twice to unit *Unit Number* with no intervening file positioning

An attempt was made to execute an **ENDFILE** statement following execution of an **ENDFILE** statement. An end-of-file cannot be output to a position after an end-of-file record. Delete the second **ENDFILE** statement.

EXECUTE_COMMAND_LINE has WAIT=.FALSE., but asynchronous execution is not supported

EXECUTE_COMMAND_LINE has **WAIT=.FALSE.**, but asynchronous execution is not supported. Check the program.

Expected decimal point in format specification

There is not decimal point in edit descriptors in **FORMAT** statements. Verify the **FORMAT** statement.

Expected integer literal constant in format specification

The form of edit description is incorrect. Possible cases include. Correct the format specification.

- The Iw.m, Zw.m, Ow.m, and Bw.m edit descriptors does not specify a value of 'm' (period specified).
- The Dw.d, Fw.d, Ew.d, ENw.d, ESw.d, and Gw.d edit descriptors does not specify a value of 'd' (period specified).
- The Ew.dEe, ENw.dEe, ESw.dEe, and Gw.dEe edit descriptors does not specify a value of 'e' (exponential character specified).
- A sign is specified for 'k' in the kP edit descriptor, and then no number is specified.
- The TLn, TRn, and Tn edit descriptor does not specify a value of 'n'.

Expected P following signed integer constant in format specification

A number with a sign is followed by a character other than character P. The signed numbers can only be specified for kP edit descriptor. Correct the format specification.

Exponent too large for Dw.d format

The exponent too large for Dw.d edit descriptor. Explicitly specify the number of exponent digits in the Ew.dEe edit descriptor. Note that changing to the Ew.dEe format will change the exponential character from D to E.

Exponent too large for Ew.d format

The exponent too large for Ew.d edit descriptor. Explicitly specify the number of exponent digits in the Ew.dEe edit descriptor.

F90_UNIX_DIR.GETCWD: Both NAME and LENNAME are not PRESENT

The **GETCWD** procedure in **F90_UNIX_DIR** module does not have a NAME and a LENNAME. Check the program.

F90_UNIX_ENV.GETARG: Value of K (*value*) is out of range 0:num

The value of K for **GETARG** procedure in **F90_UNIX_ENV** module is out of range.Check the program.

F90_UNIX_ENV.GETENV(*var*): No such environment variable

There are no environment variables specified in **GETENV** procedure for **F90_UNIX_ENV** module. Check the program.

F90_UNIX_ENV.ISATTY: LUNIT (*value*) is out of range

The logical device specification for **ISATTY** procedure in **F90_UNIX_ENV** modlue is out of range.Check the program.

F90_UNIX_ENV.SYSCONF(*value*): Not a valid sysconf name

The sysconf name specified in **SYSCONF** procedure for **F90_UNIX_ENV** module is not valid. Check the program.

F90_UNIX_ENV.SYSCONF(*value*): Result (*value*) too large for VAL

The result value of **SYSCONF** procedure for **F90_UNIX_ENV** module is too high. Check the program.

F90_UNIX_ENV.TTYNAME: LUNIT (*value*) is out of range

The logical device specification for **TTYNAME** procedure in **F90_UNIX_ENV** module is out of range.Check the program.

F90_UNIX_FILE.FSTAT: LUNIT (*value*) is out of range

The logical device specification for **FSTAT** procedure in **F90_UNIX_FILE** modlue is out of range.Check the program.

F90_UNIX_IO.FLUSH: LUNIT (*value*) is out of range

The logical device specification for **FLUSH** procedure in **F90_UNIX_IO** module is out of range.Check the program.

Field/exponent width or repeat in format specification must be non-zero

The field width or repeat factor cannot be zero. The field width or repeat factor must be a positive integer value.

File name too long

The file path name specified when opens file is too long. The file path name must be within 255 bytes.

FILE= specifier on OPEN with STATUS='SCRATCH'

SCRATCH is specified for the **STATUS** specifier in the **OPEN** statement. In this case, do not specify the **FILE** specifier. If the external file is a scratch file, delete the **FILE** specifier. Otherwise, correct the value of the **STATUS** specifier.

Floating overflow on real number input

In the execution of input statement with a real data type, a large numeric value out of the allowable range was specified. Improve the precision of a real data type, or correct the input data.

FORALL limit expression value *value* is out of range for index variable *var* type *type*

DO limit expression value *value* is out of range for index variable *var* type *type*. Change the *value* of the DO limit expression.

FORALL step expression value *value* is out of range for index variable *var* type *type*

DO step expression value *value* is out of range for index variable *var* type *type*. Change the *value* of the DO step expression.

FORALL step value is zero for index variable *var*

The **FORALL** syntax has zero steps. Non-zero.

Format specification does not end with a right parenthesis

The end of the format specification does not ending with the right parentheses. Add right parentheses at the end of the format specification.

GET argument to intrinsic RANDOM_SEED is too small (*value* elements)

The *value* of **GET** argument to intrinsic **RANDOM_SEED** is too small. Review the program.

I/O error on unit *Unit Number*: Disk quota exceeded

Writes failed because of disk quota limits at **WRITE** or **CLOSE** statements with this *unit number*. Verify the file system quota limit.

I/O error on unit *Unit Number*: Permission denied

Accesses failed because of no file permissions for this *unit number*. Verify the permission of specified file.

Illegal character '' in LOGICAL input field

For a logical type data input, a character in the input data is not acceptable. Correct the input data.**Incorrect unit for VE_FORT_MEM_BLOCKSIZE.**

An incorrect unit was specified for the environment variable **VE_FORT_MEM_BLOCKSIZE**. Verify that the unit of value specified in the environment variable **VE_FORT_MEM_BLOCKSIZE** is using "G" or "M".

Input list bigger than record length in unformatted READ on unit *Unit Number*

Input statement was attempted in excess of a record length with an unformatted input statement. Correct the unformatted input statement so that input does not exceed the record length.

Input value too large for default INTEGER type

In the execution of input statement with a default integer data type, an integer value out of the allowable range was specified. Correct the input data.

Input value too large for INTEGER(KIND=1)

In the execution of input statement with 1 byte integer data type, an integer value out of the allowable range was specified. Correct the input data.

Input value too large for INTEGER(KIND=2)

In the execution of input statement with 2 bytes integer data type, an integer value out of the allowable range was specified. Correct the input data.

Internal file overflow

The internal file in the I/O statement is overflowed. Verify that the size of scalar character variables specified in the internal file is greater than the size of output data.

Invalid character in binary integer input field

For a binary data input, a character in the input data is not acceptable. Correct the input data.

Invalid character in hexadecimal integer input field

For a hexadecimal data input, a character in the input data is not acceptable. Correct the input data.

Invalid character in integer input field

For an integer type data input, a character in the input data is not acceptable. Correct the input data.

Invalid character in octal integer input field

For an octal data input, a character in the input data is not acceptable. Correct the input data.

Invalid character in real input field

For a real type data input, a character in the input data is not acceptable. Correct the input data.

Invalid character *value* in NAMELIST input

The value of the **NAMELIST** input is not acceptable. Change the value of the character.

Invalid edit descriptor beginning with *'edit character'*

There are incorrect characters in the format specification. Correct the format specification.

Invalid edit descriptor for character i/o-list item

There is an incorrect edit descriptor in a format specification for the input/output list item of a character type. Correct the edit descriptor.

Invalid edit descriptor for integer i/o-list item

There is an incorrect edit descriptor in a format specification for the input/output list item of an integer type. Correct the edit descriptor.

Invalid edit descriptor for logical i/o-list item

There is an incorrect edit descriptor in a format specification for the input/output list item of a logical type. Correct the edit descriptor.

Invalid edit descriptor for real i/o-list item

There is an incorrect edit descriptor in a format specification for the input/output list item of a real type. Correct the edit descriptor.

Invalid edit descriptor G0.d for CHARACTER input/output item

An incorrect edit descriptor G0.d in a format specification for the input/output list item of a character type is specified. The width must be 1 or higher.

Invalid edit descriptor G0.d for INTEGER input/output item

An incorrect edit descriptor G0.d in a format specification for the input/output list item of an integer type is specified. The width must be 1 or higher.

Invalid edit descriptor G0.d for LOGICAL input/output item

An incorrect edit descriptor G0.d in a format specification for the input/output list item of a logical type is specified. The width must be 1 or higher.

Invalid exponent in real input field

For a real type data input, the exponent data in the input data is not acceptable. Correct the input data.

Invalid input for character editing

For the execution of input statement with a character data type, the form of a character input value is not acceptable. Correct the input data.

Invalid input for complex editing

For the execution of input statement with a complex data type, the form of a complex input value is not acceptable. Correct the input data.

Invalid input for integer editing

For the execution of input statement with an integer data type, the form of an integer input value is not acceptable. Correct the input data.

Invalid input for logical editing

For the execution of input statement with a logical data type, the form of a logical input value is not acceptable. Correct the input data.

Invalid input for real editing

For the execution of input statement with a real data type, the form of a real input value is not acceptable. Correct the input data.

Invalid value for ACCESS= specifier

The value of the **ACCESS** specifier in the **OPEN** statement is incorrect. The **ACCESS** specifier must be 'SEQUENTIAL', 'DIRECT', 'STREAM' or 'APPEND'.

Invalid value for ACTION= specifier

The value of the **ACTION** specifier in the **OPEN** statement is incorrect. The **ACTION** specifier must be 'READWRITE', 'READ' or 'WRITE'.

Invalid value for BLANK= specifier

The value of the **BLANK** specifier in the **OPEN** or **READ** statement is incorrect. The **BLANK** specifier must be either 'NULL' or 'ZERO'.

Invalid value for DELIM= specifier

The value of the **DELIM** specifier in the **OPEN** or **WRITE** statement is incorrect. The **DELIM** specifier must be 'NONE', 'APOSTROPHE' or 'QUOTE'.

Invalid value for FORM= specifier

The value of the **FORM** specifier in the **OPEN** statement is incorrect. The **FORM** specifier must be either 'FORMATTED' or 'UNFORMATTED'.

Invalid value for PAD= specifier

The value of the **PAD** specifier in the **OPEN** or **READ** statement is incorrect. The **PAD** specifier must be either 'YES' or 'NO'.

Invalid value for POS= specifier

The **POS** specifier for the **READ** or **WRITE** statement is incorrect. Correct the **POS** specifier value as that greater than or equal to 1.

Invalid value for POSITION= specifier

The value of the **POSITION** specifier in the **OPEN** statement is incorrect. The **POSITION** specifier must be 'ASIS', 'REWIND' or 'APPEND'.

Invalid value for RECL= specifier (must be positive)

The value of the **RECL** specifier in the **OPEN** statement is incorrect. Correct so that the value is positive integer.

Invalid value for ROUND= specifier

The value of the **ROUND** specifier in the **OPEN**, **READ** or **WRITE** statement is incorrect. The **ROUND** specifier must be 'PROCESSOR_DEFINED', 'UP', 'DOWN', 'ZERO', 'NEAREST' or 'COMPATIBLE'.

Invalid value for STATUS= specifier

The value of the **STATUS** specifier in the **OPEN** or **CLOSE** statement is incorrect. The **STATUS** specifier must be either 'KEEP' or 'DELETE'.

Invalid value for VE_FORT_MEM_BLOCKSIZE.

An incorrect value was specified for the environment variable **VE_FORT_MEM_BLOCKSIZE**. Verify that the value specified in the environment variable **VE_FORT_MEM_BLOCKSIZE** is 0 or power of 2.

Invalid value of VE_INIT_HEAP.

An incorrect value was specified for the environment variable **VE_INIT_HEAP**. Verify that the value specified in the environment variable **VE_INIT_HEAP**.

Left-hand side of assignment has duplicate vector subscript value *value* for dimension *dim*

The vector subscript for dimension *dim* on the left side duplicates in the assignment. Check the program.

Left-hand side of assignment has vector subscript *name* with duplicate value *value*

The vector subscript on the left side duplicates in the assignment. Check the program.

LEN argument (*value*) out of range 0:*bitsize* in intrinsic IBITS

The *value* of the LEN argument of intrinsic IBITS is out of range. Change the *value* of the LEN argument.

Missing length of H edit descriptor

There is no number of characters before a character H in H edit descriptor on a format specification. Correct the format specification.

Multiple assignment to scalar *var* in FORALL

Scalar *var* in the **FORALL** syntax have been assigned multiple times. Scalar *var* must be assigned only once.

Multiple assignment to scalar variable in FORALL

Scalar variables in the **FORALL** syntax have been assigned multiple times. Scalar variables must be assigned only once.

Multiple assignment to whole array *var* in FORALL

The same element of an array in the **FORALL** syntax has been assigned multiple times. The same element must be only assigned to it once.

Nested format-item-list is empty

There is no edit descriptor specified in nested of a format specification. Specify edit descriptor in nested of a format specification, or delete unnecessary nest of a format specification.

NEW file already exists

A file specified in the **OPEN** statement with **STATUS='NEW'** already exists. Check the file name and correct if it is incorrect. If file name correct, correct the value of the **STATUS** specifier in the **OPEN** statement.

NEWUNIT= specifier but no FILE= and STATUS= value is not 'SCRATCH'

The **NEWUNIT** specifier is specified for the **OPEN** statement, but the **FILE** specifier is not specified, and the value of the **STATUS** specifier is not **SCRATCH**. When opens a file on an unused unit number that is automatically chosen, specify the external file name in the **FILE** specifier or specify the scratch file with **STATUS='SCRATCH'** .Otherwise, use **UNIT** specifier instead of **NEWUNIT** specifier.

No data edit descriptor in unlimited format item

Unlimited repeat factor was specified in a format specification, but there is no data edit descriptor specified on the target nest. If you specify unlimited repeat factor, specify a data edit descriptor on the target nest. If you don't need a data edit descriptor, correct the format specification so that unlimited repeat factor are not used.

No edit descriptor following repeat factor

There is no edit descriptor following repeat factor in a format specification. Correct the format specification.

No FILE= specifier with STATUS='REPLACE' or STATUS='NEW'

REPLACE or **NEW** is specified to the **STATUS** specifier in the **OPEN** statement, but the **FILE** specifier is not specified. When specifying **REPLACE** or **NEW** to the **STATUS** specifier in the **OPEN** statement, the **FILE** specifier must also be specified. Otherwise, correct the value of the **STATUS** specifier.

No left parenthesis after unlimited repeat factor '*'

There is not specified left parenthesis after unlimited repeat factor in a format specification. If you want the unlimited repeat factor, specify left parenthesis after unlimited repeat factor. Otherwise, delete unlimited repeat factor.

No unit available for NEWUNIT= specifier

The **NEWUNIT** specifier is specified for the **OPEN** statement, but number of opens a file on an unused unit number that is automatically chosen has exceeded the limit. Close unnecessary files.

No value found in LOGICAL input field

For a logical type data input, a character in the input data is not acceptable. Correct the input data.

OPEN on connected unit *Unit Number* has different ACCESS= specifier

A different value from when the external file was connected is specified as the value of **ACCESS** specifier in the **OPEN** statement. Change the value of the **ACCESS** specifier to the same value. If you want to connect the external file with a new value, execute the **OPEN** statement after close the file.

OPEN on connected unit *Unit Number* has different ACTION= specifier

The **ACTION** specifier value of the **OPEN** statement for a device that is already connected is different than before. Change the value of the **ACTION** specifier to the same value. If you want to connect with a new value, close the device and then run the **OPEN** statement.

OPEN on connected unit *Unit Number* has different ASYNCHRONOUS= specifier

A different value from when the external file was connected is specified as the value of **ASYNCHRONOUS** specifier in the **OPEN** statement. Change the value of the **ASYNCHRONOUS** specifier to the same value. If you want to connect the external file with a new value, execute the **OPEN** statement after close the file.

OPEN on connected unit *Unit Number* has different FORM= specifier

A different value from when the external file was connected is specified as the value of **FORM** specifier in the **OPEN** statement. Change the value of the **FORM** specifier to the same value. If you want to connect the external file with a new value, execute the **OPEN** statement after close the file.

OPEN on connected unit *Unit Number* has different POSITION= specifier

A different value from when the external file was connected is specified as the value of **POSITION** specifier in the **OPEN** statement. Change the value of the **POSITION** specifier to the same value. If you want to connect the external file with a new value, execute the **OPEN** statement after close the file.

OPEN on connected unit *Unit Number* has different RECL= specifier

A different value from when the external file was connected is specified as the value of **RECL** specifier in the **OPEN** statement. Change the value of the **RECL** specifier to the same value. If you want to connect the external file with a new value, execute the **OPEN** statement after close the file.

OPEN on connected unit *Unit Number* with STATUS= specifier must have STATUS='OLD'

The external file is connected, but the value of the **STATUS** specifier in the **OPEN** statement is not **OLD**. Change the **STATUS** specifier value to **OLD**.

Out of memory

Not enough memory to run with temporary area. Check the memory size you are using and review the program.

Out of memory in intrinsic ADJUSTL

Not enough memory to run for intrinsic function **ADJUSTL** with temporary area. Check the memory size you are using and review the program.

Out of memory in intrinsic ADJUSTR

Not enough memory to run for intrinsic function **ADJUSTR** with temporary area. Check the memory size you are using and review the program.

Out of memory in intrinsic EXECUTE_COMMAND_LINE

Not enough memory to run for intrinsic function **EXECUTE_COMMAND_LINE** with temporary area. Check the memory size you are using and review the program.

Out of memory in intrinsic PACK

Not enough memory to run for intrinsic function **PACK** with temporary area. Check the memory size you are using and review the program.

Out of memory in intrinsic RESHAPE

Not enough memory to run for intrinsic function **RESHAPE** with temporary area.

Check the memory size you are using and review the program.

Out of memory in intrinsic SPREAD

Not enough memory to run for intrinsic function **SPREAD** with temporary area. Check the memory size you are using and review the program.

Out of range: substring ending position envpos is greater than length *len*

Substring ending position is greater than length. The value in the range must be specified.

Out of range: substring starting position startpos is less than 1

Substring starting position is less than 1. 1 or more must be specified.

POS argument (*value*) out of range 0:*bitsize* in intrinsic IBCLR

The *value* of the POS argument of intrinsic IBCLR is out of range. Change the *value* of the POS argument.

POS argument (*value*) out of range 0:*bitsize* in intrinsic IBITS

The *value* of the POS argument of intrinsic IBITS is out of range. Change the *value* of the POS argument.

POS argument (*value*) out of range 0:*bitsize* in intrinsic IBSET

The *value* of the POS argument of intrinsic IBSET is out of range. Change the *value* of the POS argument.

POS= specifier but unit *Unit Number* is not open for STREAM i/o

UNFORMATTED is specified for the **FORM** specifier in the **OPEN** statement. In this case, do not specify the **PAD** specifier. If the external file is an unformatted file, delete the **PAD** specifier. Otherwise, correct the value of the **FORM** specifier.

PUT argument to intrinsic RANDOM_SEED is too small (*value* elements)

The *value* of **PUT** argument to intrinsic **RANDOM_SEED** is too small. Review the program.

READ after WRITE with no intervening file positioning

An attempt was made to execute an input statement following the execution of an output statement for an external file connected as a sequential access file. Alternatively, an attempt was made to execute an input statement without a **POS** specifier following the execution of an output statement for an external file connected as a stream access. Correct so that a **REWIND** statement is executed before the input statement. Alternatively, correct so that specify a **POS** specifier in the **READ** statement for the stream file.

READ/WRITE attempted after ENDFILE on unit *Unit Number*

An attempt was made to execute an input or output statement following execution of an **ENDFILE** statement for an external file connected as a sequential access file. Alternatively, an attempt was made to execute an input or output statement immediately after an end-of-file condition. A record cannot be output to a position after an end-of-file. Correct so that a **REWIND** statement is executed before the input statement. Alternatively, when you are adding a record immediately before the end-of-file, correct so that a **BACKSPACE** statement is executed before the output statement.

RECL= specifier with ACCESS='STREAM'

STREAM is specified for the **ACCESS** specifier in the **OPEN** statement. In this case, do not specify the **RECL** specifier. If the external file is a stream file, delete the **RECL** specifier. Otherwise, correct the value of the **ACCESS** specifier.

Record longer than 2GB not supported

The record size exceeded 2 gigabytes in Sequential File Unformatted Record I/O. When this message is output at the input, check the endian format of the input data. If the endian format is Big Endian, specify environment variable **VE_FORT_UFMTENDIAN**. Otherwise, specify environment variable **VE_FORT_EXPRCW** or **VE_FORT_SUBRCW**.

Record number *Record Number* out of range

The **REC** specifier for the **READ** or **WRITE** statement is incorrect. Correct the **REC** specifier value as that greater than or equal to 1.

Reference to dangling pointer

Referring to dangling pointer. Review the program.

Reference to dangling pointer *name*

Referring to dangling pointer *name*. Review the program.

Reference to disassociated POINTER

Referring to disassociated pointer. Review the program.

Reference to disassociated POINTER *name*

Referring to disassociated pointer *name*. Review the program.

Reference to undefined POINTER

Referring to undefined pointer. Review the program.

Reference to undefined POINTER *name*

Referring to undefined pointer *name*. Review the program.

Repeat factor given for blank-interpretation edit descriptor

Do not specify the repeat factor to blank interpretation edit descriptor in a format specification. Correct the format specification.

Repeat factor given for character string edit descriptor

Do not specify the repeat factor to character string edit descriptor in a format specification. Correct the format specification.

Repeat factor given for position edit descriptor

Do not specify the repeat factor to position edit descriptor in a format specification. Correct the format specification.

Repeat factor given for rounding edit descriptor

Do not specify the repeat factor to round edit descriptor in a format specification. Correct the format specification.

Repeat factor given for sign edit descriptor

Do not specify the repeat factor to sign edit descriptor in a format specification. Correct the format specification.

Scale factor *num* out of range for d=*num*

The value of scale factor is out of range.Check the program.

SHIFT argument (*value*) out of range -*bitsize*:*bitsize* in intrinsic ISHFT

The *value* of the SHIFT argument of intrinsic ISHFT is out of range. Change the *value* of the SHIFT argument.

SHIFT argument (*value*) out of range -*size*:*size* in intrinsic ISHFTC

The *value* of the SHIFT argument of intrinsic ISHFTC is out of range. Change the *value* of the SHIFT argument.

Sign in a numeric input field not followed by any digits

For an integer or real type data input, sign in a numeric input field is not followed by any digits. Correct the input data.

SIGN= specifier must be 'PROCESSOR_DEFINED', 'PLUS' or 'SUPPRESS'

The value of the **SIGN** specifier in the **OPEN** or **WRITE** statement is incorrect. The **SIGN** specifier must be 'PROCESSOR_DEFINED', 'PLUS' or 'SUPPRESS'.

SIZE argument (*value*) out of range 1:*maxsize* in intrinsic ISHFTC

The *value* of the SIZE argument of intrinsic ISHFTC is out of range. Change the *value* of the SIZE argument.

SIZE= is not valid without ADVANCE='NO'

NO is not specified for the **ADVANCE** specifier in the **READ** statement. In this case, do not specify the **SIZE** specifier. If the **READ** statement uses advancing input, delete the **SIZE** specifier. Otherwise, specify NO to the value of the **ADVANCE** specifier.

STATUS='KEEP' is invalid for a SCRATCH file

The **STATUS** specifier in the **CLOSE** statement for **SCRATCH** file is KEEP. Correct the value of the **STATUS** specifier for the **CLOSE** statement to **DELETE**, or correct the value of the **STATUS** specifier for the **OPEN** statement to anything other than **SCRATCH**.

Subscript (*value*) out of range in input for object *objname* of NAMELIST/*namelist*/

The subscript *value* of the array is out of range in input for object of NAMELIST. Change the *value* of the subscript.

Subscript out of range for assumed-size array *name* - Access to element *value* but actual argument has only *value* elements

The subscript value of the assumed-size array *name* is out of range. Change the value of the subscript.

Subscript *rank* of *dimname* (value *value*) is out of range (*lower*:*)

The subscript *value* of the array is out of range. Change the subscript *value* of the array.

Subscript *rank* of *dimname* (value *value*) is out of range (*lower*:*upper*)

The subscript *value* of the array is out of range. Change the subscript *value* of the array.

Substring (*lower*:*upper*) out of bounds in input for object *objname* of NAMELIST/*namelist*/

The subscript *value* of the array is out of range in input for object of NAMELIST. Change the *value* of the subscript.

Substring has zero length in input for object *objname* of NAMELIST/*namelist*/

The subscript has zero length in input for object of NAMELIST. Change the *value* of the subscript.

Sub-format groups nested too deeply

Parentheses in a format specification have a nest of more than 40. The number of nests should be within 40.

The RECL= specifier must be given for DIRECT access OPEN

DIRECT is specified to the **ACCESS** specifier of the **OPEN** statement, but the **RECL** specifier is not specified. When specifying **DIRECT** to the **ACCESS** specifier in the **OPEN** statement, the **RECL** specifier must also be specified. Otherwise, correct the value of the **ACCESS** specifier.

Undefined pointer *name* used as argument to intrinsic function ASSOCIATED

An undefined pointer *name* is used as argument to intrinsic function **ASSOCIATED**. Review the program.

Undefined pointer *name* used as argument to intrinsic function EXTENDS_TYPE_OF

An undefined pointer *name* is used as argument to intrinsic function **EXTENDS_TYPE_OF**. Review the program.

Undefined pointer *name* used as argument to intrinsic function SAME_TYPE_AS

An undefined pointer *name* is used as argument to intrinsic function **SAME_TYPE_AS**. Review the program.

Undefined pointer *name* used as argument to intrinsic function STORAGE_SIZE

An undefined pointer *name* is used as argument to intrinsic function **STORAGE_SIZE**. Review the program.

Undefined pointer used as argument to intrinsic function ASSOCIATED

An undefined pointer is used as argument to intrinsic function **ASSOCIATED**. Review the program.

Undefined pointer used as argument to intrinsic function EXTENDS_TYPE_OF

An undefined pointer is used as argument to intrinsic function **EXTENDS_TYPE_OF**. Review the program.

Undefined pointer used as argument to intrinsic function SAME_TYPE_AS

An undefined pointer is used as argument to intrinsic function **SAME_TYPE_AS**. Review the program.

Undefined pointer used as argument to intrinsic function STORAGE_SIZE

An undefined pointer is used as argument to intrinsic function **STORAGE_SIZE**. Review the program.

Undefined polymorphic pointer *name* used as argument to intrinsic function ASSOCIATED

An undefined polymorphic pointer *name* used as argument to intrinsic function **ASSOCIATED**. Review the program.

Undefined polymorphic pointer *name* used as argument to intrinsic function EXTENDS_TYPE_OF

An undefined polymorphic pointer *name* used as argument to intrinsic function **EXTENDS_TYPE_OF**. Review the program.

Undefined polymorphic pointer *name* used as argument to intrinsic function SAME_TYPE_AS

An undefined polymorphic pointer *name* used as argument to intrinsic function **SAME_TYPE_AS**. Review the program.

Undefined polymorphic pointer *name* used as argument to intrinsic function STORAGE_SIZE

An undefined polymorphic pointer *name* used as argument to intrinsic function **STORAGE_SIZE**. Review the program.

Undefined polymorphic pointer used as argument to intrinsic function ASSOCIATED

An undefined polymorphic pointer used as argument to intrinsic function **ASSOCIATED**. Review the program.

Undefined polymorphic pointer used as argument to intrinsic function

EXTENDS_TYPE_OF

An undefined polymorphic pointer used as argument to intrinsic function **EXTENDS_TYPE_OF**. Review the program.

Undefined polymorphic pointer used as argument to intrinsic function SAME_TYPE_AS

An undefined polymorphic pointer used as argument to intrinsic function **SAME_TYPE_AS**. Review the program.

Undefined polymorphic pointer used as argument to intrinsic function STORAGE_SIZE

An undefined polymorphic pointer used as argument to intrinsic function **STORAGE_SIZE**. Review the program.

Unexpected exponent for G0 edit descriptor

Zero was specified to width for Gw.dEe edit descriptor. If you want the width to be zero, correct to Gw.d edit descriptor. Otherwise, specify the positive value to the width.

Unexpected subscript for object *objname* of NAMELIST/*namelist*/

The subscript is an unexpected for object of NAMELIST. Change the *value* of the subscript.

Unit number *Unit Number* out of range

The value of the **UNIT** specifier is incorrect. The **UNIT** specifier must be an integer value from 0 to 2147483647 or a value returned to the **NEWUNIT** specifier in the **OPEN** statement.

Unit *Unit Number* is not connected

The specified external unit is not connected to file. Correct so that the file is opened before executing the input/output statement for the specified external unit.

Unit *Unit Number* is not connected for DIRECT i/o

An attempt was made to execute the sequential or stream access input/output statement on a file connected as a direct access file. Correct to the direct access input/output statement, or correct so that the file is connected by the sequential or stream access file.

Unit *Unit Number* is not connected for FORMATTED i/o

An attempt was made to execute a formatted input/output statement on a file connected as an unformatted file. Correct to the unformatted input/output statement, or correct so that the file is connected by a formatted file.

Unit *Unit Number* is not connected for READ action

An attempt was made to output to an external file for which input only is permitted. Correct the external unit number if it is wrong. Otherwise, connect that external unit number to a file which accepts output with an **OPEN** statement.

Unit *Unit Number* is not connected for SEQUENTIAL i/o

An attempt was made to execute a sequential access input/output statement on a file connected as a direct access file. Correct to the direct access input/output statement, or correct so that the file is connected by a sequential access file.

Unit *Unit Number* is not connected for UNFORMATTED i/o

An attempt was made to execute an unformatted input/output statement on a file connected as a formatted file. Correct to the formatted input/output statement, or correct so that the file is connected by an unformatted file.

Unit *Unit Number* is not connected for WRITE action

An attempt was made to input from an external file for which output only is permitted. Correct the external unit number if it is wrong. Otherwise, connect that external unit number to a file which accepts input with an **OPEN** statement.

Unit *Unit Number* is not connected on OPEN with STATUS='OLD' and no FILE= specifier

OLD is specified for the **STATUS** specifier in the **OPEN** statement, but the **FILE** specifier is not. When specifying OLD for the **STATUS** specifier in the **OPEN** statement, the **FILE** specifier must also be specified. Otherwise, correct the value of the **STATUS** specifier.

VALUE argument (*value*) to intrinsic ATOMIC_ADD is out of range

The value of argument to intrinsic function ATOMIC_ADD is out of range.Check the program.

VALUE argument (*value*) to intrinsic ATOMIC_AND is out of range

The value of argument to intrinsic function ATOMIC_AND is out of range.Check the program.

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_ADD is out of range

The value of argument to intrinsic function ATOMIC_FETCH_ADD is out of range.Check the program.

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_AND is out of range

The value of argument to intrinsic function ATOMIC_FETCH_AND is out of range.Check the program.

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_OR is out of range

The value of argument to intrinsic function ATOMIC_FETCH_OR is out of range.Check the program.

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_XOR is out of range

The value of argument to intrinsic function ATOMIC_FETCH_XOR is out of range.Check the program.

VALUE argument (*value*) to intrinsic ATOMIC_OR is out of range

The value of argument to intrinsic function ATOMIC_OR is out of range.Check the program.

VALUE argument (*value*) to intrinsic ATOMIC_XOR is out of range

The value of argument to intrinsic function ATOMIC_XOR is out of range.Check the program.

VALUES argument to intrinsic DATE_AND_TIME is too small (*value* elements)

The *value* of VALUES argument to intrinsic **DATE_AND_TIME** is too small. Review the program.

Value *value* of KIND argument to OMP_SET_SCHEDULE is out of range 1:4

The value of KIND argument to OMP_SET_SCHEDULE is out of range. The value must be an integer value from 1 to 4.

Value *value* of MAX_LEVELS argument to OMP_SET_MAX_ACTIVE_LEVELS is negative

The value specified for the OMP_SET_MAX_ACTIVE_LEVELS is negative. Must be a positive integer.

Value *value* of NUM_THREADS argument to OMP_SET_NUM_THREADS is greater than maximum num

The value specified for the OMP_SET_NUM_THREADS exceeds the maximum value. Must be in the range.

Value *value* of NUM_THREADS argument to OMP_SET_NUM_THREADS is not positive

The value specified for the OMP_SET_NUM_THREADS is not positive. Must be a positive integer.

***var* has not been assigned a branch target label**

Var has not been assigned a branch target label. Review the program.

Vector subscript for rank *rank* of *name* has extent *value* instead of *value*

The vector subscript size of the rank dimension is incorrect. Change the value of the vector subscript.

WRITE operation failed on unit *Unit Number*: Disk quota exceeded

Writes failed because of disk quota limits at **WRITE** statements with this *unit number*. Verify the file system quota limit.

Zero repeat factor in list-directed input

For the r*c form of a list-directed input value, the repeat factor r is zero. Correct the repeat factor r to 1 or higher.

Zero stride value for subscript *num* of *name*

The stride value for subscript is zero. Change the stride value.

12.3 Other Runtime Error

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.

Chapter13 Troubleshooting

13.1 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 "Invalid #line directive" occurs.

Directive of preprocessors such as "#if, #include" is used. Please compile with **-fpp**.

The error "Cannot find module : ..." occurs.

A module was used, but the compiler could not find the module file (*.mod). Please confirm whether a module file exists in the directory by which a compiler searches a module file. Please refer to "1.6 Searching Module Files" about the directory a compiler searches.

The error "not a valid module information file" occurs.

There is a possibility that a module file was compiled by an old compiler or is broken. Please remake a module file (*.mod).

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 E 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 module file, 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 Module Files ", "1.7 Searching files included by INCLUDE line or #include directive" and "1.8 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.

```
$ nfort 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.

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 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
  (...)
```

I want to confirm whether they are object file for VE3.

Please execute "/opt/nec/ve/bin/nreadelf -h" that specified the object file as an argument of command. When the last digit output in the line of "Flags:" is "0", it show that a file is an object file for VE1; when it is "1", it show that a file is an object file for VE3. In the following example, the last digit output in the line of "Flags:" is "1", so it show that a file is an object file for VE3.

```
$ /opt/nec/ve/bin/nreadelf -h a.o
ELF Header:
  (...)
  Version:                           0x1
  Start of program headers:          0 (bytes into file)
  Start of section headers:          720 (bytes into file)
  Flags:                             0x10101
  (...)
```

When linking OpenMP and automatic parallelized program, which of -fopenmp and -mparallel should I specify?

Please specify either **-fopenmp** or **-mparallel**.

```
$ nfort -c -mparallel a.f90
$ nfort -c -fopenmp b.f90
$ nfort -fopenmp a.o b.o
```

When specifying -fcheck, compilation time becomes so long.

It becomes long because check code is inserted at compilation. Please specify **-fcheck** to only the source file which includes procedure which need check.

When specifying -fcheck, execution time becomes so long.

It becomes long because check code is executed. Please specify **-fcheck** to only the source file which includes procedure which need check.

When specifying -ftrace, execution time becomes so long.

It becomes long because extra routines for getting performance information are executed at entrance/exit of procedures 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”.

I want to preprocess Fortran program.

Please compile the program with **-fpp**.

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.

The message “Internal error detected -- please report.” is output.

When compilation is not stopped at the message output, the compiler recover the error and continues compiling. In this case, created object file can be used without problems. When compilation is stopped, please contact us from the NEC support portal site.

The following message is output though ALLOCATE or DEALLOCATE statement is not in a loop.

```
vec(181): Allocation obstructs vectorization.
vec(182): Deallocation obstructs vectorization.
```

This message is output when the compiler needed to allocate and deallocate an area at execution to realize language specification of Fortran. It may occur when passing argument or return value at inlining a procedure.

I want to know about difference between -bss and -save.

In case of variable of SAVE attribute, initialized value in a routine is return value of called last time. In case of **-bss**, it is not guaranteed.

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**.

I want to create a position-independent executable with the option -fpie or -fPIE.

Creation of a position-independent executable is not supported.

The error "Too many elements in array" occurs.

The size of the array allocated by the ALLOCATE statement, or the size of the array allocated by the DIMENSION statement/attribute, exceeds 1TiB. Please review the size of array.

Note: The upper limit of the array size is checked with 1TiB at compilation, but the memory size of VE is 48GB. Therefore, if you try to allocate the array larger than the memory size of VE, it occurs "Out of memory" at run-time.

A .L file is not generated When compiling a module source file.

A L files is not generated for module source files that do not contain module procedures according to its specifications.

When building a program that includes multi-stage dependencies such as a.out->foo.so->bar.so, the following link error occurs.

```
/opt/nec/ve/bin/nld: warning: libbar.so, needed by ./libfoo.so, not found
 (try using -rpath or -rpath-link)
./libfoo.so: undefined reference to `bar'
```

It is a GNU Linker specification from which nld is derived. The nld links SX-Aurora TSUBASA objects on Linux/x86_64, so it works with cross linker. Since Cross Linker is not always the same as the actual execution environment, nld ignores the **-rpath** option and **RPATH** set in the shared library. Please specify **-Wl,-rpath-link,*library-path***.

When -mparallel was specified, the following warning occurred.

```
/opt/nec/ve/bin/nld: warning: libnfort.so.2, needed by libxxx.so, not found (try
using -rpath or -rpath-link)
```

libnfort.so is a library that contains Fortran runtime routines and is required whenever linking a non-parallel version of a Fortran program. When linking a parallel version of a Fortran program (with **-mparallel** or **-fopenmp**), libnfort_m.so is always required instead. The compiler automatically specifies "-lnfort" for non-parallel and "-lnfort_m" for parallel at the time of linking. The warning is that libxxx.so is created non-parallel and requires a non-parallel libnfort.so, but "-lnfort" is required because "-lnfort" is not specified when -mparallel is specified.

Since unexpected problems may occur when running the libxxx.so, it is recommended to additionally specify **-mparallel** or **-fopenmp** when creating (linking) the libxxx.so to reference libnfort_m.so..

13.2 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: libnfort.so.2: cannot open shared object file: No such file or directory” is output at execution.

Please install the package “nec-nfort-shared” and “nec-nfort-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 “2.2 Environment Variables Referenced During Execution” 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 **VE_ADVANCEOFF**=YES. An execution time may increase substantially to stop the advance control. Please take care it.

```
$ export VE_ADVANCEOFF=YES
```

I want to output the debug write result from buffer at exception occurrence.

Please call the **FLUSH** statement after the **WRITE** statement.

```
SUBROUTINE SUB()
  INTEGER :: U, X, A(20)

  OPEN(NEWUNIT=U, FILE='debug.log', STATUS='replace')

  CALL SUB1(X)

#ifdef DEBUG
  WRITE(U, *) 'X=', X
  FLUSH(U)
#endif

  WRITE(*,*) A(1000)
END
```

I want to confirm whether use uninitialized variable or not.

It may be checked by detecting an exception to compile with **-minit-stack**=snan and execute with the environment variable **VE_INIT_HEAP**=SNAN for double precision floating-point type variables. For single precision floating-point type variables, specify snanf and SNANF instead of snan and SNAN. This approach can be used only if the variable is floating-point type. Please refer to “2.2 Environment Variables Referenced During Execution” and “3.6 Debugging Options”.

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 with the environment variable **VE_INIT_HEAP**=ZERO. Correction of a program is recommended to resolve a potential problem. Please refer to “2.2 Environment Variables Referenced During Execution” and “3.6 Debugging Options”.

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 or decrease the stack usage.

The limit of stack size can be increased by setting the environment variable

OMP_STACKSIZE. Please refer to “2.2 Environment Variables Referenced During Execution”.

```
$ export OMP_STACKSIZE=2G
```

- The used stack can be decreased to specify the **-mno-stack-arrays**. Please note that the execution time can be increased by specifying **–mno-stack-arrays**.

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.

I want to conform the stack size required to run the program.

There is no way to find out the required stack size because you will not know it until you try it.

13.3 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.

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.

The routine name which name is “$” and number as ‘$1’ is displayed in FTRACE output. What is it?

It is created by compiler to do vectorization and parallelization.

13.4 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/nfort --version
nfort (NFORT) 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 details, 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**).

13.5 Troubleshooting for SX-ACE compiler migration

The -ew option is specified.

Check the program to see if it applies to the following:

(1) When you are using intrinsic procedures by specific-name, modify it to a double-precision or generic-name.

(2) Modify the type declarations and constants in the program as shown in the following.

FORTRAN90/SX Compiler	Vector Engine Compiler
INTEGER*2	INTEGER*8
INTEGER*4	INTEGER*8
INTEGER(KIND=2)	INTEGER(KIND=8)
INTEGER(KIND=4)	INTEGER(KIND=8)
LOGICAL*1	LOGICAL*8
LOGICAL*4	LOGICAL*8
LOGICAL(KIND=1)	LOGICAL(KIND=8)
LOGICAL(KIND=4)	LOGICAL(KIND=8)
REAL*4	REAL*8
REAL(KIND=4)	REAL(KIND=8)
COMPLEX*8	COMPLEX*16
COMPLEX(KIND=4)	COMPLEX(KIND=8)
Constants 1.23E1	Constants 1.23D1
Constants 1.23_4	Constants 1.23_8

(3) Specify both options **-fdefault-real**=8 and **-fdefault-integer**=8 when compiling. This compiler option is not required when you modified program to specify the kind type in a type declaration.

The -A dbl option is specified.

Please do one of the following.

(1) Modify the type declarations and constants in the program as shown in the following.

FORTRAN90/SX Compiler	Vector Engine Compiler
REAL*4	REAL*8
REAL*8	REAL*16

FORTRAN90/SX Compiler	Vector Engine Compiler
REAL(KIND=4)	REAL(KIND=8)
REAL(KIND=8)	REAL(KIND=16)
COMPLEX*8	COMPLEX*16
COMPLEX*16	COMPLEX*32
COMPLEX(KIND=4)	COMPLEX(KIND=8)
COMPLEX(KIND=8)	COMPLEX(KIND=16)
Constants 1.23E1	Constants 1.23D1
Constants 1.23D1	Constants 1.23Q1
Constants 1.23_4	Constants 1.23_8
Constants 1.23_8	Constants 1.23_16

(2) Specify both options **-fdefault-real**=8 and **-fdefault-double**=16 when compiling. This compiler option is not required when you modified program to specify the kind type in a type declaration.

The -A dbl4 option is specified.

Please do one of the following.

(1) Modify the type declarations and constants in the program as shown in the following.

FORTRAN90/SX Compiler	Vector Engine Compiler
REAL*4	REAL*8
REAL(KIND=4)	REAL(KIND=8)
COMPLEX*8	COMPLEX*16
COMPLEX(KIND=4)	COMPLEX(KIND=8)
Constants 1.23E1	Constants 1.23D1
Constants 1.23_4	Constants 1.23_8

(2) Specify options **-fdefault-real**=8 when compiling. This compiler option is not required when you modified program to specify the kind type in a type declaration.

The -A dbl8 option is specified.

Please do one of the following.

(1) Modify the type declarations and constants in the program as shown in the following.

FORTRAN90/SX Compiler	Vector Engine Compiler
REAL*8	REAL*16
REAL(KIND=8)	REAL(KIND=16)
COMPLEX*16	COMPLEX*32
COMPLEX(KIND=8)	COMPLEX(KIND=16)
Constants 1.23D1	Constants 1.23Q1
Constants 1.23_8	Constants 1.23_16

(2) Specify option **-fdefault-double**=16 when compiling. This compiler option is not required when you modified program to specify the kind type in a type declaration.

The environment variable F_UFMTADJUST=TYPE2 is specified when inputting the binary file.

Specify the environment variable VE_FORT_UFMTADJUST, when inputting binary file　that specified and created by the environment variable F_UFMTADJUST.

Inputting binary file created with SX-ACE.

Specify the environment variable **VE_FORT_UFMTENDIAN**, when inputting binary file created with SX-ACE.

Chapter14 VE1/VE3 Compatibility

14.1 Executables Compatibility

VE1/VE3 machine can execute the following executables generated by VE1/VE3 compiler/assembler/linker.

Executables	VE1 Machine	VE3 Machine
Executables for VE1	✓	✓
Executables for VE3	-	✓

(✓) Can be executed (-) Cannot be executed

14.2 Changes of Search Path

Search path of module files, files included by **INCLUDE** line or **#include** directive and libraries are changed as follows:

(1) Searching Module Files

VE1	VE3
Directory on which each input source file is	(Same as VE1)
Directories specified by **-module**	(Same as VE1)
Current directory	(Same as VE1)
Directories specified by **-I**	(Same as VE1)
Subdirectory named “include” under the directory specified by **-B**	(Same as VE1)
Directories specified by the environment variable **NFORT_INCLUDE_PATH**	(Same as VE1)
Directory specified by **-isystem**	(Same as VE1)
/opt/nec/ve/nfort/<*version-number*>/include	(Same as VE1)
/opt/nec/ve/include (*1)	/opt/nec/ve3/include (*1)

(*1) When **-isysroot** is enabled, subdirectory named “include” under the directory specified by **-isysroot**.

(2) Searching files included by **INCLUDE** line or **#include** directive

VE1	VE3
Directory on which each input source file is	(Same as VE1)
Directories specified by **-module**	(Same as VE1)
Current directory	(Same as VE1)
Directories specified by **-I**	(Same as VE1)
Subdirectory named “include” under the directory specified by **-B**	(Same as VE1)
Directories specified by the environment variable **NFORT_INCLUDE_PATH**	(Same as VE1)
Directory specified by **-isystem**	(Same as VE1)
/opt/nec/ve/nfort/<*version-number*>/include	(Same as VE1)
/opt/nec/ve/include (*1)	/opt/nec/ve3/include (*1)

(*1) When **-isysroot** is enabled, subdirectory named “include” under the directory specified by **-isysroot**.

(3) Searching Libraries

VE1	VE3
Directories specified by **-L**	(Same as VE1)
Directories specified by **-B**	(Same as VE1)
Directories specified by the environment variable **NFORT_LIBRARY_PATH**	(Same as VE1)
/opt/nec/ve/nfort/<*version-number*>/lib	/opt/nec/ve3/nfort/<*version-number*>/lib
Directories specified by the environment variable **VE_LIBRARY_PATH**	(Same as VE1)
/opt/nec/ve/lib/gcc	/opt/nec/ve3/lib/gcc
/opt/nec/ve/lib	/opt/nec/ve3/lib

14.3 Changes of Compiler Options

VE1/VE3 changes the defaults of compiler options as follows:

VE1	VE3
-march=ve1	**-march**=ve3
-mfp16-format=none	**-mfp16-format**=ieee

14.4 Half-Precision Floating-Point Type

VE3 can generate and execute object files using half-precision floating point. Object files using half-precision floating point cannot be generated or executed on VE1.

14.4.1 Format of Half-Precision Floating-Point Type

The format of half-precision floating-point type is determined by **-mfp16-format**=*type* and whether or not half-precision floating-point is used in the program.

Use Half-Precision Floating-Point or Not	-mfp16-format=*type*		
	none	**ieee**	**bfloat**
Not Use	none	none	none
Use	none	binary16	bfloat16

14.4.2 Mixing binary16 and bfloat16

When both binary16 and bfloat16 object files are mixed, an object file, executable file or shared library cannot be generated.

14.5 Notice

- VE3 executables cannot be executed on VE1.
- In VE1, it is not possible to generate or execute object files using half-precision floating-point.
- The command ngprof cannot output a performance information when executing the VE1 executables on VE3 and output "gmon.out".
- Unable to generate static libraries, shared libraries, or executables with a mix of VE1 and VE3 object files. The following error occurs when linking.

```
/opt/nec/ve/bin/nld: a.o: this object cannot use on VE3.
/opt/nec/ve/bin/nld: failed to merge target specific data of file a.o
```

- When using the "traceback" function of the compiler with VE1 binaries, please ensure that version 5.0.1 or later is used. Additionally, if using the "traceback" function of MPI, please ensure that MPI version 3.4.0 or later is also used in conjunction with the compiler. Otherwise, the generated traceback may not be outputted correctly.

Chapter15 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) 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
```

(4) 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.

(5) The NAMELIST output is changed to new form since version 3.0.8. If you want to NAMELIST output form of version 3.0.7 or earlier, set "NO" to environment variable **VE_FORT_NML_REPEAT_FORM**.

```
$ export VE_FORT_NML_REPEAT_FORM=NO
```

(6) The compiler outputs the following info message since version 3.5.0, when the argument type of intrinsic procedures SYSTEM_CLOCK is other than INTEGER(KIND=8). This message recommends INTEGER(KIND=8) as the argument type of intintrinsic procedures SYSTEM_CLOCK, but does not necessarily require any program changes. Even if you do not modify the program, it will not affect the program execution.

```
The arguments to intrinsic subroutine SYSTEM_CLOCK are of type INTEGER, but it
is recommended that they should be of type INTEGER(int64)
```

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*.

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/nfort/version)
as	The path of assembler command (default: <veroot>/bin/nas)
fcom	The path of Fortran compiler (default: <system>/libexec/fcom)
ld	The path of linker command (default: <veroot>/bin/nld)
fpp	The path of Fortran preprocessor command (default: <system>/libexec/fpp)
fc_pre_options **fc_post_options**	The Compiler options. The options are specified in the following order. <fc_pre_options> <user-specified-options> <fc_post_options>
as_pre_options **as_post_options**	The Assembler options. The options are specified in the following order. <as_pre_options> <user-specified-options> <as_post_options>
ld_pre_options **ld_post_options**	The Linker options. The options are specified in the following order. <ld_pre_options> <user-specified-options> <ld_post_options>
startfile	The startup file.
endfile	The startup file. The file is specified at the tail of 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:

```
fc_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/nfort/X.X.X
```

  When the configuration file is specified by **-cf**. The configuration file name is ve.conf here.

```
$ nfort -cf=ve.conf test.f90
```

- Change the using compiler.

  Set the value to 'fcom' when only the used compiler is changed.

```
fcom: /foo/ve/nfort/X.X.X/libexec/fcom
```

  When the configuration file is specified by **-cf**. 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 compiler options, compiler directives, and environment variables referred at the execution between SX compilers and compilers for the Vector Engine.

B.1 NEC Fortran 2003 Compiler Options

B.1.1 Overall Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-Caopt	**-O4**
-Chopt	**-O3**
-Cvopt	**-O2**
-Csopt	**-O2 -mno-vector**
-Cvsafe	**-O1**
-Cssafe	**-O1 -mno-vector**
-Cnoopt	**-O0**
-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**
-mod | -Nmod	none
-o *file-name*	**-o** *file-name*
-size_t32	none
-size_t64	none Note: Always effective.

NEC Fortran 2003 Compiler	Vector Engine Compiler
-syntax	**-fsyntax-only**
-Nsyntax	**-fno-syntax-only**
-tm *directory-name*	none
-to *directory-name*	none
-verbose	**-v**
-Nverbose	none

B.1.2 Vector/Scalar Optimization Options

NEC Fortran 2003 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**
-Onoextendreorder	none
-Oignore_volatile	**-fignore-volatile**
-Onoignore_volatile	**-fno-ignore-volatile**
-Oiodo	**-marray-io**
-Onoiodo	**-mno-array-io**
-Omove	**-fmove-loop-invariants-unsafe**
-Onomovediv	**-fmove-loop-invariants**
-Onomove	**-fno-move-loop-invariants**
-Ooverlap	**-fnamed-alias**
-Onooverlap	**-fnamed-noalias**
-Oreorderrange=bblock	**-msched-insns**

NEC Fortran 2003 Compiler	Vector Engine Compiler
-Ounroll	**-floop-unroll**
-Ounroll=*n*	**-floop-unroll** **-floop-unroll-max-times=***n* Note: Specify two at the same time.
-Onounroll	**-fno-loop-unroll**
-dir { vec | novec }	none
-ipa	**-fipa**
-Nipa	**-fno-ipa**
-math { errchk | noerrchk }	none
-math { inline | noinline }	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-vecgtor-depencendy-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,chgpwr	**-mvector-power-to-explog** **-mvector-power-to-sqrt** Note: Specify two at the same time.

NEC Fortran 2003 Compiler	Vector Engine Compiler
-pvctl,collapse	**-floop-collapse**
-pvctl,nocollapse	**-fno-loop-collapse**
-pvctl { compress | nocompress }	none
-pvctl,cond_mem_opt	**-mvector-merge-conditional**
-pvctl,nocond_mem_opt	**-mno-vector-merge-conditional**
-pvctl { conflict | noconflict }	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**
-pvctl,noloopchg	**-fno-loop-interchange**
-pvctl,loopcnt=*n*	**-floop-count**=*n*
-pvctl,lstval	none
-pvctl,nolstval	none
-pvctl,matmul	**-fmatrix-multiply**
-pvctl,nomatmul	**-fno-matrix-multiply**
-pvctl { neighbors | noneighbors }	none
-pvctl,nodep	**-fivdep**
-pvctl,on_adb[=*category***]**	none
-pvctl,outerunroll=*n*	**-fouterloop-unroll** **-fouterloop-unroll-max-times**=*n* Note: Specify two at the same time.
-pvctl,outerunroll_lim=*n*	none

NEC Fortran 2003 Compiler	Vector Engine Compiler
-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,vwork={ static | stack | hybrid }	none
-pvctl,vworksz=*n*	none
-salloc	**-mstack-arrays**
-Nsalloc	**-mno-stack-arrays**
-v	**-mvector**
-Nv	**-mno-vector**
-xint	**-mno-vector-iteration**
-Nxint	**-mvector-iteration**

B.1.3 Inlining Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-dir { inline | noinline }	none
-pi,auto	**-finline-functions**
-pi,max_depth=*n*	**-finline-max-depth=***n*
-pi,max_size=*n*	**-finline-max-function-size=***n*
-pi,proc_size=*n*	none
-pi,times=*n*	**-finline-max-times=***n*

B.1.4 Parallelization Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-dir { par | nopar }	none
-Pauto	**-mparallel**
-Pmulti	none
-Popenmp	**-fopenmp**
-Pstack	none
-Pstatic	**-bss**
-pvctl,for[=*n*]	none Note: Parallelization schedule can be controlled by **-mschedule-static** etc.
-pvctl,by=*n*	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

NEC Fortran 2003 Compiler	Vector Engine Compiler
-adv { on | off }	none

NEC Fortran 2003 Compiler	Vector Engine Compiler
-Nadv	none
-mask { flovf | flunf | fxovf | inv | inexact | zdiv }	none Note: It can be controlled by the envir onment variable VE_FPE_ENABL E.
-mask { setall | nosetall | setmain }	none
-prec_complex_division	none
-Nprec_complex_division	none
-stkchk | -Nsckchk	none

B.1.6 Language Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-defacto_associated	none
-Ndefacto_associated	none
-default_double_size	**-fdefault-double**=*n*
-default_real_size	**-fdefault-real**=*n*
-default_integer_size	**-fdefault-integer**=*n*
-extend_source	**-fextend-source**
-fixed	**-ffixed-form**
-free	**-ffree-form**
-f2003 **-f2008** **-f95**	**-std**={ **f2003** | **f2008** | **f95** }
-ignore_directive	none
-Nignore_directive	none
-small_integer | -Nsmall_integer	none

B.1.7 Performance Measurement Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-acct	**-proginf**
-Nacct	**-no-proginf**
-ftrace	**-ftrace**
-Nftrace	**-no-ftrace**
-p	**-p**
-Np	none

B.1.8 Debug Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-check	**-fcheck**=keyword
-init stack={ zero | nan | *0xXXXX* **}**	**-minit-stack={ zero | snan | snanf |** *0xXXXX* **}**
-mtrace [basic]	**-mmemory-trace**
-mtrace full	**-mmemory-trace-full**
-Nmtrace	none
-traceback	**-traceback**
-Ntraceback	none

B.1.9 Preprocessor Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-D*name*[=_def_**]**	**-D*name*[=**_def_**]**
-E	**-E**
-EP	none
-Ep	**-fpp**

NEC Fortran 2003 Compiler	Vector Engine Compiler
-NE	**-nofpp**
-H	none
-I *directory-name*	**-I** *directory-name*
-M	**-M**
-U*name*	**-U***name*
-Wp,*option-string*	**-Wp,***option-string*
-ts *directory-name*	none

B.1.10 List Output Options

NEC Fortran 2003 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
-R { transform | notransform }	none

B.1.11 Message Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-O { fullmsg | infomsg | nomsg }	none

NEC Fortran 2003 Compiler	Vector Engine Compiler
-pi { fullmsg | infomsg | nomsg }	**-fdiag-inline**={ 2 | 1 | 0 }
-pvctl { fullmsg | infomsg | nomsg }	**-fdiag-parallel**={ 2 | 1 | 0 } **-fdiag-vector**={ 2 | 1 | 0 }
-w all	**-Wall**
-w none	**-w**
-w { info | noinfo }	none
-w extension	**-Wextension**
-w noextension	**-Wno-extension**
-w { observe | noobserve }	none
-w obsolescent	**-Wobsolescent**
-w noobsolescent	**-Wno-obsolescent**
-w { unreffed | nounreffed }	none
-w {unused | nounused }	none

B.1.12 Assembler Option

NEC Fortran 2003 Compiler	Vector Engine Compiler
-Wa,*option-string*	**-Wa,*option-string***

B.1.13 C Compiler Option

NEC Fortran 2003 Compiler	Vector Engine Compiler
-Wc,*option-string*	none

B.1.14 Linker Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-L *directory-name*	**-L** *directory-name*

NEC Fortran 2003 Compiler	Vector Engine Compiler
-l*library-name*	**-l***library-name*
-Wl,*option-string*	**-Wl,***option-string*

B.1.15 Directory Options

NEC Fortran 2003 Compiler	Vector Engine Compiler
-YI,*directory-name*	none
-YL,*directory-name*	none
-YM,*directory-name*	none
-YS,*directory-name*	none
-Ya,*directory-name*	none
-Yf,*directory-name*	none
-Yl,*directory-name*	none
-Yp,*directory-name*	none

B.2　FORTRAN90/SX Compiler

B.2.1　f90/sxf90 command Options

FORTRAN90/SX Compiler	Vector Engine Compiler
-Chopt	**-O3**
-Cvopt	**-O2**
-Csopt	**-O2 –mno-vector**
-Cvsafe	**-O1**
-Cssafe	**-O1 –mno-vector**
-Cdebug	**-O0 -g**
-c	**-c**
-Nc	none

FORTRAN90/SX Compiler	Vector Engine Compiler
-cf *strings*	**-cf=*strings***
-clear	**-clear**
-D*name*[=*def*]	**-D*name*[=*def*]**
-da	none
-dC	**-fcheck=none**
-dD	none
-dP	none
-dR	**-fcheck=none**
-dW	none Note: **-dW** is always effective.
-dw	none Note: **-dw** is always effective.
-ea	none
-eC	**-fbounds-check** or **-fcheck=bounds**
-eD	none
-eP	none
-eR	**-fbounds-check** or **-fcheck=bounds** Note: Only the range of array subscripts is checked.
-eW	none
-ew	none Note: See Section 13.5 for details of migration.
-EP	none
-Ep	**-fpp**
-NE	**-nofpp**
-f2003	none Note: Fortran 2003 features are available by default.
-f2003 { cbind | nocbind }	none
-f2003 { cptr_derive | cptr_i8 }	none

FORTRAN90/SX Compiler	Vector Engine Compiler
-f2003 { opt_ieee | noopt_ieee }	none
-Nf2003	none
-f0	**-ffixed-form**
-f3	**-ffixed-form –fextend-source**
-f4	**-ffree-form**
-f5	**-ffree-form –fextend-source**
-ftrace	**-ftrace**
-Nftrace	**-no-ftrace**
-G { global | local }	none
-g	**-g**
-gv	none
-gw	none
-Ng	**-g0**
-I *directory-name*	**-I** *directory-name*
-L *directory-name*	**-L** *directory-name*
-l*library-name*	**-l***library-name*
-o *file-name*	**-o** *file-name*
-Pauto	**-mparallel**
-Pmulti	none
-Popenmp	**-fopenmp**
-Pstack	none
-Pstatic	**-bss**
-p	**-p**
-Np	none
-pi argconsis={noexp|safe|unsafe}	none
-pi auto	**-finline-functions**
-pi noauto	none
-pi exp=*procedure-name*	none

FORTRAN90/SX Compiler	Vector Engine Compiler
-pi noexp=*procedure-name*	none
-pi expin={*file-name***|***directory***}**	**-finline-file=***file-name* or **-finline-directory=***directory* Note: **-finline-functions** option is needed.
-pi { fullmsg | infomsg | nomsg }	**-fdiag-inline**={ **2 | 1 | 0** }
-pi { incdir | noincdir }	none
-pi line=*n* Note: *n* is the number of lines of the source code.	**-finline-max-function-size=***n* Note: *n* is the amount of intermediate representations for a function**.** **-finline-functions** option is needed.
-pi { modout | nomodout }	none
-pi nest=*n*	**-finline-max-depth=***n* Note: **-finline-functions** option is needed.
-pi rexp=*function*	none
-Npi	**-fno-inline-functions**
-R0	none
-R1	none
-R2	none
-R3	none
-R4	none
-R5	**-report-diagnostics -report-format**
-S	**-S**
-NS	none
-size_t32	none
-size_t64	none Note: **-size_t64** is always effective.
-sx8 | -sx8r | -sx9 | -sxace	none
-to *directory-name*	none

FORTRAN90/SX Compiler	Vector Engine Compiler
-ts *directory-name*	none
-U*name*	**-U***name*
-V Note: Continue the compilation process.	**--version** Note: Display the version and exit.
-NV	none
-verbose	**-v**
-Nverbose	none
-Wa,*option-strings*	**-Wa,***option-strings*
-Wc,*option-strings*	none
-Wf,*option-strings* Note: See the following sections for detailed options.	none
-Wl,*option-strings*	**-Wl,***option-strings*
-Wp,*option-strings*	**-Wp,***option-strings*
-w	**-w**
-Nw	**-Wall**
-Yf,*directory-name*	none
-Yl,*directory-name*	none
-Yp,*directory-name*	none

B.2.2 f90/sxf90 Detailed Options for optimization

FORTRAN90/SX Compiler	Vector Engine Compiler
-ai | -Nai	none
-fusion	**-floop-fusion**
-Nfusion	**-fno-loop-fusion**
-i { errchk | noerrchk }	none
-O { aryinq | noaryinq }	none
-O chg	**-fassociative-math** or **-faggressive-associative-math**

FORTRAN90/SX Compiler	Vector Engine Compiler
-O nochg	**-fno-associative-math** Note: **-faggressive-associative-math** optimize more aggressive than **-fassociative-math.**
-O { compass | nocompass }	none
-O darg	**-fargument-alias**
-O nodarg	**-fargument-noalias**
-O div	**-freciprocal-math**
-O nodiv	**-fno-reciprocal-math**
-O extendreorder	**-msched-interblock**
-O reorderrange=bblock	**-msched-insns**
-O { if | noif }	none
-O iodo	**-marray-io**
-O noiodo	**-mno-array-io**
-O infomsg	none
-O move	**-fmove-loop-invariants-unsafe**
-O nomovediv	**-fmove-loop-invariants**
-O nomove	**-fno-move-loop-invariants**
-O overlap	**-fnamed-alias**
-O nooverlap	**-fnamed-noalias**
-O { shapeprop | noshapeprop }	none
-O unroll	**-floop-unroll**
-O unroll=*n*	**-floop-unroll** **-floop-unroll-max-times=***n* Note: Specify two at the same time.
-O nounroll	**-fno-loop-unroll**
-O wkary_opt	**-mstack-arrays**
-O nowkary_opt	**-mno-stack-arrays**

FORTRAN90/SX Compiler	Vector Engine Compiler
-O { zlpchk | nozlpchk }	none
-prob_dir *directory-name*	none
-prob_file *file-name*	none
-prob_generate	none
-prob_use	none

B.2.3　f90/sxf90 Detailed Options for vectorization and parallelization

FORTRAN90/SX Compiler	Vector Engine Compiler
-common { global | local }	none
-moddata { global | local }	none
-ompctl { condcomp | nocondcomp }	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-vecgtor-depencendy-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

FORTRAN90/SX Compiler	Vector Engine Compiler
-pvctl chgpwr	**-mvector-power-to-explog** **-mvector-power-to-sqrt** Note: Specify two at the same time.
-pvctl chgtanh	none
-pvctl cncall=*routine-name*	none
-pvctl collapse	**-floop-collapse**
-pvctl nocollapse	**-fno-loop-collapse**
-pvctl { compress | nocompress }	none
-pvctl cond_mem_opt	**-mvector-merge-conditional**
-pvctl nocond_mem_opt	**-mno-vector-merge-conditional**
-pvctl { conflict | noconflict }	none
-pvctl divloop	none
-pvctl nodivloop	**-mwork-vector-kind**=**none**
-pvctl expand=*n*	**-floop-unroll-complete**=*n*
-pvctl noexpand	**-fno-loop-unroll-complete**
-pvctl { farouter | nofarouter }	none
-pvctl for[=*n***]**	none Note: Parallelization schedule can be controlled by **-mschedule-static** etc.
-pvctl by=*n*	none Note: Parallelization schedule can be controlled by **-mschedule-static** etc.
-pvctl { fullmsg | infomsg | nomsg }	**-fdiag-parallel**=**{ 2 | 1 | 0 }** **-fdiag-vector**=**{ 2 | 1 | 0 }** Note: Specify two at the same time.
-pvctl { ifopt | noifopt }	none
-pvctl inner	**-mparallel-innerloop**
-pvctl noinner	**-mno-parallel-innerloop**
-pvctl listvec	**-mlist-vector**

FORTRAN90/SX Compiler	Vector Engine Compiler
-pvctl nolistvec	**-mno-list-vector**
-pvctl loopchg	**-floop-interchange**
-pvctl noloopchg	**-fno-loop-interchange**
-pvctl loopcnt=*n*	**-floop-count**=*n*
-pvctl lstval	none
-pvctl nolstval	none
-pvctl matmul	**-fmatrix-multiply**
-pvctl nomatmul	**-fno-matrix-multiply**
-pvctl matmulblass	none
-pvctl { neighbors | noneighbors }	none
-pvctl nodep	**-fivdep**
-pvctl on_adb[=*category***]**	none
-pvctl outerstrip	**-mparallel-outerloop-strip-mine**
-pvctl noouterstrip	**-mno-parallel-outerloop-strip-mine**
-pvctl outerunroll=*n*	**-fouterloop-unroll** **-fouterloop-unroll-max-times**=*n* Note: Specify two at the same time.
-pvctl outerunroll_lim=*n*	none
-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
-pvctl shape=*n*	none
-pvctl split	**-floop-split**
-pvctl nosplit	**-fno-loop-split**
-pvctl { vchg | novchg }	none
-pvctl vecthreshold=*n*	**-mvector-threshold**=*n*

FORTRAN90/SX Compiler	Vector Engine Compiler
-pvctl verrchk	**-mvector-intrinsic-check**
-pvctl noverrchk	**-mno-vector-intrinsic-check**
-pvctl { vlchk | novlchk }	none
-pvctl vregs=*n*	none
-pvctl vsqrt	**-mvector-sqrt-instruction**
-pvctl novsqrt	**-mno-vector-sqrt-instruction**
-pvctl vwork={ static | stack | hybrid }	none
-pvctl vworksz=*n*	none
-reserve *n*	none
-tasklocal { macro | micro }	none
-v	**-mvector**
-Nv	**-mno-vector**

B.2.4 f90/sxf90 Other Detailed Options

FORTRAN90/SX Compiler	Vector Engine Compiler
-A { dbl | dbl4 | dbl8 | idbl | idbl4 | idbl8 }	**-A idbl : -fdefault-real=8 -fdefault-double=16** **-A idbl4 : -fdefault-real=8** **-A idbl8 : -fdefault-double=16** Note: See Section 13.5 for details of migrating other options.
-acct	**-proginf**
-Nacct	**-no-proginf**
-adv { on | off }	none
-Nadv	none
-compatimod	none
-const_ext | -Nconst_ext	none
-cont	**-fassume-contiguous**
-Ncont	**-fno-assume-contiguous**

FORTRAN90/SX Compiler	Vector Engine Compiler
-dblprecision | -Ndblprecision	none
-dir { vec | par | debug }	none
-dir { novec | nopar | nodebug }	none
-dollar | -Ndollar	none
-esc | -Nesc	none
-G | -NG	none
-init stack={ zero | nan | *0xXXXX* **}**	**-minit-stack={ zero | nan |** *0xXXXX* **}**
-init heap={zero | nan | *0xXXXX* **}**	none Note: It can be controlled by the environment variable **VE_INIT_HEAP**.
-K { a | Na }	none
-K { b | Nb }	none
-L { stdout | nostdout | filename=*file-name* **}**	**-report-file={ stdout |** *file-name* **}** Note: The default is **-Lnostdout**.
-L { eject | noeject }	none
-L fmtlist	**-report-format**
-L nofmtlist	none
-L { inclist | noinclist }	none
-L { map | nomap }	none
-L mrgmsg	none
-L sepmsg	**-report-diagnostics**
-L objlist	**-assembly-list**
-L noobjlist	none
-L { source | nosource }	none
-L { summary | nosummary }	none
-L { transform | notransform }	none

FORTRAN90/SX Compiler	Vector Engine Compiler
-NL	none
-M { zdiv | flovf | fxovf | inv | inexact }	none Note: It can be controlled by the environment variable **VE_FPE_ENABLE**.
-M { setall | setmain }	none
-msg b	**-Wobsolescent**
-msg nb	**-Wno-obsolescent**
-msg { d | nd }	none
-msg { f | nf }	none Note: **nf** is always effective.
-msg { o | no }	none
-msg { w | nw }	none. Note: **nw** is always effective.
-P { a | b | c | d | e | f | h | i | l | p | t | x | z }	none
-P { b | nb }	none
-P { c | nc }	none
-P { d | nd }	none
-P { e | ne }	none
-P f	**-nofpp**
-P nf	none Note: **nf** is always effective.
-P h	none Note: **h** is always effective.
-P nh	**-ff90-sign**
-P { i | ni }	none
-P { l | nl }	none
-P { p | np }	none

FORTRAN90/SX Compiler	Vector Engine Compiler
-P { t | nt }	none Note: **nt** is always effective.
-P { x | nx }	none Note: **x** is always effective.
-P { z | nz }	none
-ptr { byte | word }	none Note: **byte** is always effective.
-s | -Ns	none
-stmtid | -Nstmtid	none
-w { double16 | rdouble16 }	none
-xint	**-mno-vector-iteration**
-Nxint	**-mvector-iteration**

B.3 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.4 Environment Variables

SX Compiler	Vector Engine Compiler
F_PROGINF	**VE_PROGINF**
F_TRACEBACK	**VE_TRACEBACK**
F_EXPRCW	**VE_FORT_EXPRCW**
F_FMTBUF	**VE_FORT_FMTBUF**
F_NORCW	**VE_FORT_NORCW**
F_PAUSE	**VE_FORT_PAUSE**

SX Compiler	Vector Engine Compiler
F_PARTRCW	**VE_FORT_PARTRCW**
F_SETBUF	**VE_FORT_SETBUF**
F_UFMTADJUST=TYPE1	**VE_FORT_UFMTADJUST=INT,LOG**
F_UFMTADJUST=TYPE2	**VE_FORT_UFMTADJUST=ALL**
F_UFMTENDIAN	**VE_FORT_UFMTENDIAN**
F_FF*n*	**VE_FORT***n*

B.5 Other Library

-use can be used instead of **USE** statement.

SX Compiler	Vector Engine Compiler
CALL ABORT()	USE F90_UNIX CALL ABORT()
RESULT = ACCESS(NAME,MODE)	USE F90_UNIX_FILE CALL ACCESS(NAME,AMODE,RESULT) Note: MODE(CHARACTER) was changed to AMODE(INTEGER). See Section 11.3.5 for details of AMODE(INTEGER).
RESULT = ALARM(SECONDS,HANDLER)	USE F90_UNIX_PROC CALL ALARM(SECONDS,HANDLER,RESULT,ERRNO)
RESULT = CHDIR(NAME)	USE F90_UNIX_DIR CALL CHDIR(NAME,RESULT)
RESULT = CHMOD(NAME,MODE)	USE F90_UNIX_FILE CALL CHMOD(PATH,AMODE,RESULT) Note: MODE(CHARACTER) was changed to AMODE(INTEGER). See Section 11.3.5 for details of AMODE(INTEGER).
CALL FLUSH(UNIT)	FLUSH(UNIT)
RESULT = FORK()	USE F90_UNIX_PROC CALL FORK(RESULT,ERRNO)
CALL FREE(PTR)	USE F90_UNIX CALL FREE(PTR)

SX Compiler	Vector Engine Compiler
RESULT = FSTAT(UNIT,BUFF)	USE F90_UNIX_FILE CALL FSTAT(UNIT,BUFF,RESULT)
CALL GETARG(POS,VALUE)	USE F90_UNIX CALL GETARG(POS,VALUE)
RESULT = GETCWD(DIRNAME)	USE F90_UNIX_DIR CALL GETCWD(DIRNAME,ERRNO=RESULT)
CALL GETENV(NAME,VALUE)	USE F90_UNIX CALL GETENV(NAME,VALUE)
RESULT = GETGID()	USE F90_UNIX RESULT = GETGID()
CALL GETLOG(NAME)	USE F90_UNIX_ENV CALL GETLOGIN(NAME)
RESULT = GETPID()	USE F90_UNIX RESULT = GETPID()
RESULT = GETUID()	USE F90_UNIX RESULT = GETUID()
RESULT = HOSTNM(NAME)	USE F90_UNIX_ENV CALL GETHOSTNAME(NAME,RESULT)
RESULT = IARGC()	USE F90_UNIX RESULT = IARGC()
RESULT = ISATTY(UNIT)	USE F90_UNIX_ENV CALL ISATTY(UNIT,RESULT,ERRNO)
RESULT = LINK(PATH1,PATH2)	USE F90_UNIX_DIR CALL LINK(PATH1,PATH2,RESULT)
RESULT = LSTAT(FILE,BUFF)	USE F90_UNIX_FILE CALL LSTAT(FILE,BUFF,RESULT)
PTR = MALLOC(SIZE)	USE F90_UNIX PTR = MALLOC(SIZE)
RESULT = RENAME(FROM,TO)	USE F90_UNIX_DIR CALL RENAME(FORM,TO,RESULT)
CALL SLEEP(SECONDS)	USE F90_UNIX_PROC CALL SLEEP(SECONDS)
RESULT = STAT(FILE,BUFF)	USE F90_UNIX_FILE CALL STAT(FILE,BUFF,RESULT)
RESULT = SYSTEM(COMMAND)	USE F90_UNIX_PROC CALL SYSTEM(COMMAND,RESULT,ERRNO)

SX Compiler	Vector Engine Compiler
RESULT = TIME()	USE F90_UNIX_ENV CALL TIME(RESULT)
RESULT = TTYNAM(UNIT)	USE F90_UNIX_ENV CALL TTYNAME(UNIT,RESULT,ERRNO)
RESULT = UNLINK(PATH)	USE F90_UNIX_DIR CALL UNLINK(PATH,RESULT)
RESULT = WAIT(I)	USE F90_UNIX_PROC CALL WAIT(I,ERRNO=RESULT)

B.6 Implementation-Defined Specifications

B.6.1 Data Types

Type	SX Compiler		Vector Engine Compiler	
	Kind Type Parameter	**Data Type** (*1)	**Kind Type Parameter**	**Data Type**
integer	1 (*2)	1-byte integer	1	1-byte integer
integer	2	2-byte integer	2	2-byte integer
integer	4	4-byte integer (default integer type)	4	4-byte integer (default integer type)
integer	8	8-byte integer	8	8-byte integer
real	4	4-byte real (default real type)	4	4-byte real (default real type)
real	8	8-byte real	8	8-byte real
real	16	16-byte real	16	16-byte real
complex	4	(4,4)-byte complex (default complex type)	4	(4,4)-byte complex (default complex type)
complex	8	(8,8)-byte complex	8	(8,8)-byte complex
complex	16	(16,16)-byte complex	16	(16,16)-byte complex
logical	1	1-byte logical	1	1-byte logical
logical	4	4-byte logical (default logical type)	4	4-byte logical (default logical type)

Type	SX Compiler		Vector Engine Compiler	
	Kind Type Parameter	Data Type (*1)	Kind Type Parameter	Data Type
logical	8	8-byte logical	8	8-byte logical
character	1	character (default character type)	1	character (default character type)
character	2 (*3)	character	none	

(*1) For FORTRAN90/SX compiler, "Data Type" declaration can be changed by specifying the compiler option.

(*2) Not available with FORTRAN90/SX Compiler.

(*3) Not available with NEC Fortran2003 Compiler

B.6.2 Specifications

Items	FORTRAN90/SX Compiler	NEC Fortran 2003 Compiler	Vector Engine Compiler
Nesting level of files included by **INCLUDE** line	-	20	63
Rank of an array	7	31	31
Number of continuation lines	99	511	1023
Length of a name	63	199	199

B.6.3 Intrinsic Procedures

Intrinsic Procedures	SX Compiler	Vector Engine Compiler
SYSTEM_CLOCK	The starting point of the acquisition time is the start of the program.	The starting point of the acquisition time is 00:00 on January 1, 1970, Coordinated Universal Time (UTC).

Appendix C Compiler Directive Conversion Tool

This appendix describes the tool for converting from the SX compiler directive to the Vector Engine.

C.1 nfdirconv

Name:

nfdirconv

SYNOPSIS:

nfdirconv [OPTION...] [FILE | DIRECTORY]...

DESCRIPTION:

This tool converts the nfort/ncc/nc++ 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.

Option	Description
-q, --quiet	Do not report about conversion.
-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)]

  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.f90
program main
    integer s
!CDIR NOVECTOR
    do i=1, 1000
      s = s + i
    enddo
    print*,s
end program

$ nfdirconv sample.f90
sample.f90: line   3: converted 'NOVECTOR' to 'novector' (Converted)

$ cat sample.f90
program main
    integer s
!NEC$ novector
    do i=1, 1000
      s = s + i
    enddo
    print*,s
end program
```

Example2: When a directory is specified.

Take the following directory as an example.

dir/

+ Makefile
+ sample1.c
+ sample2.c
+ subdir/
    + Makefile
    + sample3.c

```
$ nfdirconv dir
dir/sample1.f90: line   5: converted 'loopcnt=5' to 'loop_count(5)' (Converted)
dir/sample2.f90: 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.

```
$ nfdirconv -r dir
dir/sample2.f90: line   5: converted 'nodep' to 'ivdep' (Substitute)
dir/sample1.f90: line  16: converted 'loopcnt=5' to 'loop_count(5)' (Converted)
dir/subdir/sample3.f90: 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	Vector Engine Compiler
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**
array(*c1*[,*c2*…])	(Removed/Obsolescent)
arraycomb	(Removed/Obsolescent)
assert	(Removed/Obsolescent)
assoc	**assoc**
noassoc	**noassoc**
assume	**assume**
noassume	**noassume**
atomic	**atomic**
cncall	**cncall**
collapse	**collapse**
compress	(Removed/Obsolescent)
nocompress	(Removed/Obsolescent)
concur	**concurrent**

SX Compiler	Vector Engine Compiler
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**
end arraycomb	(Removed/Obsolescent)
end parallel sections	(Removed/Obsolescent)
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	**always_inline**
inner	**inner**
noinner	**noinner**
listvec	**list_vector**
nolistvec	**nolist_vector**
loopchg	**interchange**
noloopchg	**nointerchange**
loopcnt=*n*	**loop_count**(*n*)
lstval	**lstval**
nolstval	**nolstval**
move	**move_unsafe**

SX Compiler	Vector Engine Compiler
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 do	**parallel do**
parallel do private(*identifier*)	**parallel do private**(*identifier*)
parallel sections	(Removed/Obsolescent)
section	(Removed/Obsolescent)
select(*keyword*)	**select_concurrent** **select_vector**
shape	(Removed/Obsolescent)
shortloop	**shortloop**
skip	(Removed/Obsolescent)
sparse	**sparse**
nosparse	**nosparse**
split	(Remained)
nosplit	(Remained)
sync	(Remained)
nosync	**nosync**
threshold	(Removed/Obsolescent)
othreshold	(Removed/Obsolescent)
traceback	(Remained)
unroll=*n*	**unroll**(*n*)
nounroll	**nounroll**
unshared	(Removed/Obsolescent)

SX Compiler	Vector Engine Compiler
vecthreshold	**vector_threshold**(*n*)
vector	**vector**
novector	**novector**
verrchk	(Remained)
noverrchk	(Remained)
vlchk	(Removed/Obsolescent)
ovlchk	(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)
zcheck	(Removed/Obsolescent)
nozcheck	(Removed/Obsolescent)

C.4 Notes

- 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 sxf90/sxf03/sxcc/sxc++ 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.
- **BEGIN**/**END** Directive are treated as unsupported compiler directive.

Appendix D File I/O Analysis Information

This appendix describes the File I/O Analysis Information.

D.1 Output Example

Output when the value “DETAIL” is set in the environment variable **VE_FORT_FILEINF.**

```
                  ******  File Information  ******
Unit No.   : 10
File Name  : fort.10
Named      : YES
Current Directory     : /usr/uhome/xxxxxxxx
TMPDIR                : /tmp

I/O Exec. Count       :        READ        WRITE        OPEN      CLOSE    INQUIRE
                                  1            1           0          1          0
                             REWIND    BACKSPACE     ENDFILE
                                  1            0           0
                               WAIT        FLUSH
                                  0            0

Format                :          FORMATTED    Access               :    SEQUENTIAL
Blank(OPEN)           :               NULL    Blank(READ)          :          NULL
Delim(OPEN)           :               NONE    Delim(WRITE)         :          ----
Pad(OPEN)             :                YES    Pad(READ)            :           YES
Decimal(OPEN)         :              POINT    Decimal(R/W)         :         POINT
Sign(OPEN)            :          PROCESSOR    Sign(WRITE)          :     PROCESSOR
Round(OPEN)           :          PROCESSOR    Round(R/W)           :     PROCESSOR
Asynchronous          :                 NO    Encoding             :       DEFAULT
Position              :             REWIND
Recl  (Byte)          :              65536
File Size  (Byte)     :                 13    File Descriptor      :             5
File System Type      :     NFS(0x00006969)   Open Mode            :     READWRITE
Terminal Assignment   :                 NO    Shrunk File          :           YES
Max File Size(Byte)   :                600

I/O Buffer Size (KByte)    :         512
Record Buffer Size (Byte)  :       65536

                                 Total(In/Out)          Input          Output
Total Data Size (Byte)     :               25,            13,              12
Max   Data Size (Byte)     :                              13,              12
Min   Data Size (Byte)     :                              13,              12
```

```
Ave    Data Size (Byte)      :          12,          13,          12
Transfer Rate (KByte/sec)    :      18.789,      19.261,      18.303

                        Total(In/Out/Aux)       Input       Output
Real Time (sec)              :    0.004284,    0.000659,    0.000640
User Time (sec)              :    0.002874,    0.000062,    0.000129

Environment Variable List :
```

D.2 Description of items

Unit No.

External unit identifier number.

File Name

The file name output here is a name specified in the **FILE** specifier or during preconnection; the name does not include the home directory or current directory. For SCRATCH files, file names assigned by the system are output.

Named

Whether the file is a named file.

Current Directory

The directory name currently in operation.

TMPDIR

The directory name the SCRATCH file was created. This information is output only for SCRATCH files.

I/O Exec Count

The execution count of each I/O statement. For direct access, information about REWIND, BACKSPACE and ENDFILE is not output.

Format

The value of the **FORM** specifier.

Access

The value of the **ACCESS** specifier.

Blank (OPEN)

The value of the **BLANK** specifier of the **OPEN** statement. This information is output only for FORMATTED.

Blank (READ)

The value of the **BLANK** specifier of the **READ** statement. For no **READ** statement, '----' is output. When the different value is specified in the **READ** statement, “MIXED” is output. This information is output only for FORMATTED.

Delim (OPEN)

The value of the **DELIM** specifier of the **OPEN** statement. This information is output only for FORMATTED.

Delim (WRITE)

The value of the **DELIM** specifier of the **WRITE** statement. For no **WRITE** statement, ‘----‘ is output. When the different value is specified in the **WRITE** statement, “MIXED” is output. This information is output only for FORMATTED.

Pad (OPEN)

The value of the **PAD** specifier of the **OPEN** statement. This information is output only for FORMATTED.

Pad (READ)

The value of the **PAD** specifier of the **READ** statement. For no **READ** statement, ‘----‘ is output. When the different value is specified in the **READ** statement, “MIXED” is output. This information is output only for FORMATTED.

Decimal (OPEN)

The value of the **DECIMAL** specifier of the **OPEN** statement. This information is output only for FORMATTED.

Decimal (R/W)

The value of the **DECIMAL** specifier of the **READ**/**WRITE** statement. For no **READ**/**WRITE** statement, ‘----‘ is output. When the different value is specified in the **READ**/**WRITE** statement, “MIXED” is output. This information is output only for FORMATTED.

Sign (OPEN)

The value of the **SIGN** specifier of the **OPEN** statement. This information is output only for FORMATTED.

Sign (WRITE)

The value of the **SIGN** specifier of the **WRITE** statement. For no **WRITE** statement, ‘----‘ is output. When the different value is specified in the **WRITE** statement, “MIXED” is output. This information is output only for FORMATTED.

Round (OPEN)

The value of the **ROUND** specifier of the **OPEN** statement. This information is output only for FORMATTED.

Round (R/W)

The value of the **ROUND** specifier of the **READ**/**WRITE** statement. For no **READ**/**WRITE** statement, '----' is output. When the different value is specified in the **READ**/**WRITE** statement, "MIXED" is output. This information is output only for FORMATTED.

Asynchronous

The value of the **ASYNCHRONOUS** specifier.

Encoding

The value of the **ENCODING** specifier of the **OPEN** statement. This information is output only for FORMATTED.

Position

The value of the **POSITION** specifier of the **OPEN** statement. For direct access, this information is not output.

Recl

The value of the **RECL** specifier of the **OPEN** statement in bytes. The default value is output when the **RECL** specifier is not specified. For stream access, this information is not output.

Max Record No.

The maximum record number actually input and output. This is not the maximum record number derived from the file size. This information is output only for direct access.

File Size

The size of the file in bytes at closing. This value also contains the record control word appended by program for sequential access output.

File Descriptor

The value of the file descriptor.

File System Type

The file system to which the file belongs.

Open Mode

The mode in which the file was opened.

Terminal Assignment

Whether the file is connected to a terminal.

Shrunk File

Whether the file shrinkage function was executed. The file shrinkage function releases the remaining area, when the file size at closing is smaller than the file size at opening or the maximum file size is reached during program execution. This information is output only for sequential access.

Max File Size

The maximum file size in bytes during program execution. This information is output only when the shrunk file indicates "YES". This is useful information when trying to decide on I/O buffer size.

I/O Buffer Size

The size of an I/O buffer allocated for I/O in kilo bytes.

Record Buffer Size

The size of a record buffer allocated for I/O in bytes.

Total Data Size

The total amount of transferred data in bytes. The size is output in the order of total input and output, total input, total output. The record control word appended by program during sequential access is excluded from these quantities.

Max Data Size

The maximum input and output size of transferred data in bytes. The size is output in the order of input, output.

Min Data Size

The minimum input and output size of transferred data in bytes. The size is output in the order of input, output.

Ave Data Size

The average size of transferred data in bytes. The size is output in the order of total input and output, total input, total output. This information shows whether the file I/O is small or large.

Transfer Rate

The file transfer speed in kilo bytes. The value is obtained by dividing the Total Data Size by elapsed time. This information is output only when "DETAIL" is set in **VE_FORT_FILEINF**.

Real Time

Elapsed time. This information is output only when "DETAIL" is set in **VE_FORT_FILEINF**.

User Time

User time. This information is output only when "DETAIL" is set in

VE_FORT_FILEINF.

Environment Variable List

A list of the environment variable. Only an effective environment variable output by alphabetical order. This information is output only when "DETAIL" is set in **VE_FORT_FILEINF**.

Appendix E　Change Notes

The following changes are done from the previous version (Rev.31 Jun.2023 released).

- Add the descriptions for the treatment of fixed/free format and preprocessor usage for each Fortran source files' suffix in Section 1.5.
- Change the description of **VE_FORT_EXPRCW**, **VE_FORT_NORCW**, **VE_FORT_PARTRCW**, **VE_FORT_SUBRCW** and **VE_FORT_UFMTENDIAN_NOVEC** in Section 2.2.
- Add Example5 in the description of **VE_FORT_UFMTENDIAN** in Section 2.2.
- Add the notice when using the "traceback" function on VE3 in Chapter 14.

Index

$

$ 109

&

& 110

@

@*file-name* 33

1

1-byte Integer 113
1-byte Logical 119

2

2-byte Integer 114

4

4-byte Integer 114
4-byte Logical 119

8

8-byte Integer 114
8-byte Logical 119

A

Accuracy degration 7
advance_gather 58
always_inline 58, 79
Argument Association 109
Arithmetic exception
  Accuracy degration 7
  Division by zero 7
  Floating-point overflow 7
  Floating-point underflow 7
  Invalid operation 7
  Using Traceback Information 8
  Vector instruction 8
Arithmetic Exception Mask 8
Arithmetic Exceptions 6
Arithmetic IF Statement 105
Array Complement 109
-assembly-list 52
ASSIGN statement 112
assigned GO TO statement 112
assoc 58
assume 58
atomic 58
Automatic inlining 79
Automatic Parallelization 84
automatic vectorization 68

B

-B 54
-Bdynamic 52
Binary Type 121
Boz-literal-constant 111
-bss 47
-Bstatic 53

C

-c 32
C_PTR 147
-cf 32
Character Type 119
-clear 32
cncall 59
Code Generation Module 97
COMMON Statement 100
Compares absolute values 71
Compiler Directive Conversion Tool 341
Compiler Directives 58
COMPLEX DOUBLE PRECISION Statement 100

COMPLEX DOUBLE Statement 100
Complex Double-Precision Type 117
COMPLEX QUADRUPLE PRECISION Statement 100
COMPLEX QUADRUPLE Statement 100
Complex Quadruple-Precision Type 118
Complex Single-Precision Type 117
Complex Type 117
Compression 72
Computed GO TO Statement 104
concurrent 59
Conditional Parallelization Using Dependency Test 84
Conditional Parallelization Using Threshold Test 84
Conditional Vectorization 73
Configuration file 312
Cross-file Inlining 81
Currency Symbol $ 109
-cxxlib 52

D

-D 51
DATA Statement 101
Data Types 113
dependency_test 59
Diagnostic List 91
DIMENSION Statement 101
Division by zero 7
-dM 51
DOUBLE COMPLEX Statement 101
DOUBLE PRECISION Statement 102
DOUBLE Statement 101
Double-Precision Type 115

E

-E 51
Environment Variables 10
EQUIVALENCE Statement 103
Expansion 72
Explicit inlining 79
Expressions 110
Extended Free Source Form 110

F

-faggressive-associative-math 33
-fargument-alias 33
-fargument-noalias 33
-fassociative-math 33
-fassume-contiguous 33
-fbounds-check 45
-fcheck 45
-fcopyin-intent-out 34
-fcse-after-vectorization 34
-fdefault-double 47
-fdefault-integer 47
-fdefault-real 47
-fdiag-inline 49
-fdiag-parallel 50
-fdiag-vector 50
-fextend-source 47
-ffast-formatted-io 34
-ffast-math 34
-ffast-math-check 34
-ffixed-form 48
-ffree-form 48
-fignore-asynchronous 34
-fignore-induction-variable-overflow 34
-fignore-volatile 34
-finline-abort-at-error 42
-finline-copy-arguments 42
-finline-directory 42
-finline-file 43
-finline-functions 43
-finline-max-depth 43
-finline-max-function-size 43
-finline-max-times 43
-finline-suppress-diagnostics 43
-finstrument-functions 44
-fintrinsic-modules-path 54
-fivdep 34
-fivdep-omp-worksharing-loop 34

Fixed Source Form......110
Floating-Point Data......114
Floating-point overflow 7
Floating-point underflow 7
-floop-collapse 34
-floop-count 35
-floop-fusion 35
-floop-interchange......35
-floop-normalize......35
-floop-split......35
-floop-strip-mine 35
-floop-unroll 35
-floop-unroll-complete 35
-floop-unroll-complete-nest 35
-floop-unroll-max-times 36
-fmatrix-multiply 36
-fmax-continuation-lines 48
-fmove-loop-invariants......36
-fmove-loop-invariants-if......36
-fmove-loop-invariants-unsafe 36
-fmove-nested-loop-invariants-outer 36
-fnamed-alias 36
-fnamed-noalias......36
-fnamed-noalias-aggressive......36
-fno-inline-directory......42
-fno-inline-file 43
-fopenmp 41
Forced Loop Parallelization 85
forced_collapse 59
forced-parallelization 62
Format List 92
FORMAT Statement 103
Formatted Records 125
Fortran
    arguments 153
Fortran 2018 Extensions 132
FORTRAN77 POINTER Statement 106
-fouterloop-unroll 37
-fouterloop-unroll-max-size 37
-fouterloop-unroll-max-times......37
-fpic 44
-fPIC 44
-fpp 51
-fpp-name 51
-fprecise-math 37
-frealloc-lhs...... 48
-frealloc-lhs-array 48
-frealloc-lhs-scalar 48
-freciprocal-math...... 37
-freorder-logical-expression 37
-freplace-loop-equation 37
-freplace-matmul-to-matrix-multiply 37
-fsyntax-only...... 32
-ftrace 44
FUNCTION Statement 103

G

-g 45
gather_reorder...... 59

H

H edit descriptor...... 112
Half-Precision Floating-Point Type 308
--help...... 54
Hexadecimal Type...... 120
Hollerith Assignment Statement 112
Hollerith Relational Expression...... 111
Hollerith Type...... 111, 120
HOME...... 10

I

-I...... 51
Implementation-Defined Specifications...... 113
IMPLICIT Statement...... 105
inf...... 121
inline......59, 79
inline directive 79
inline_complete 59, 79
Inlining...... 79
Inlining Module...... 96

inner....60
Integer Type....113
interchange....60
Intrinsic Procedures.... 124, 158
Invalid operation.... 7
-isysroot....51
-isystem....51
Iteration....70
ivdep....60

J

-J....54

L

-l....53
-L....53
Language-Mixed Programming....138
LD_LIBRARY_PATH....12
Linking....157
list_vector....60
Logical Operator....111
Logical Type....119
LOOP....86
loop_count....60
loop_count_test....60
lstval....60

M

-M....52
Macro Operations....69
  Compares absolute values....71
  Compression....72
  Expansion....72
  Iteration....70
  Maximum values and minimum values....70
  Product....70
  Search....72
  Sum or inner product....70
-march....44
-marray-io....37
-masync-io.... 48
Matrix Multiply Library.... 205
Maximum Array Rank.... 111
Maximum values and minimum values.... 70
-mconditional-index-test.... 37
-mcreate-threads-at-startup.... 41
memory block.... 124
Messages.... 248
-mfp16-format.... 44
-mgenerate-il-file.... 43
-minit-stack.... 46
-mlist-vector.... 38
-mmemory-trace.... 46
-mmemory-trace-full.... 46
-mno-stack-arrays.... 38
-module.... 54
move.... 60
move_unsafe.... 60
-mparallel.... 41
-mparallel-innerloop.... 41
-mparallel-omp-routine.... 41
-mparallel-outerloop-strip-mine.... 42
-mparallel-sections.... 42
-mparallel-threshold.... 42
-mread-il-file.... 43
-mretain.... 38
-msched.... 38
-mschedule-chunk-size.... 42
-mschedule-dynamic.... 42
-mschedule-runtime.... 42
-mschedule-static.... 42
-mstack-arrays.... 38
-muse-mmap.... 38
-mvector.... 38
-mvector-advance-gather.... 39
-mvector-advance-gather-limit.... 39
-mvector-assignment-threshold.... 39
-mvector-dependency-test.... 39
-mvector-floating-divide-instruction.... 39
-mvector-fma.... 39

-mvector-intrinsic-check 39
-mvector-iteration 39
-mvector-iteration-unsafe...............................40
-mvector-loop-count-test 40
-mvector-low-precise-divide-function.............40
-mvector-merge-conditional 40
-mvector-neighbors 40
-mvector-packed 40
-mvector-power-to-explog.............................40
-mvector-power-to-sqrt 40
-mvector-reduction 41
-mvector-shortloop-reduction 41
-mvector-sqrt-instruction 41
-mvector-threshold..41
-mwork-vector-kind 41, 68

N

NAMELIST Input Format.............................131
NAMELIST Output Format 131
NaN...121
neighbors 61
nfdirconv 341
NFORT_COMPILER_PATH 10
NFORT_INCLUDE_PATH 10
NFORT_LIBRARY_PATH................................10
NFORT_PROGRAM_PATH..............................11
noadvance_gather 58
noassoc...58
noassume...58
noconcurrent 59
nofma 61
nofuse 61
noinline 59, 79
noinner 60
nointerchange...60
nolist_vector...60
nolstval 60
nomove...60
noouterloop_unroll 62
nopacked_vector..62
-noqueue 55
noshortloop_reduction................................. 63
nosparse... 64
-nostartfiles 53
-nostdinc 52
-nostdlib 53
nosync 61
nounroll 64
novector 64
novob... 64
novovertake 64
novwork 65

O

-o 32
-O... 33
Octal Type 121
OMP_NUM_THREADS 12
OMP_STACKSIZE 12
Optimizations 66
Optimizing Mask Operations 68
Option List 91
options 61
Outer Loop Strip-mining 73
outerloop_unroll 62

P

-p 44
-P 52
Packed vector instructions 75
packed_vector... 62
parallel do.. 62
PARALLEL LOOP 86
PARALLEL MASTER 86
Parallelization 84
Parallelization of inner Loops 84
PARAMETER Statement 105
Partial Vectorization 68
PATH.. 11
PAUSE statement..................................... 112

-pedantic-errors 50
-pg 44
POINTER Statement 106
Preconnection 129
Predefined Macro 122
-print-file-name 54
-print-prog-name 54
Product 70
-proginf..45
-pthread..42
pvreg 62

Q

QUADRUPLE PRECISION Statement..............108
QUADRUPLE Statement..............................108
Quadruple-Precision Type 116

R

-rdynamic..53
Real Type 114
Relational Operator...................................110
-report-all..50
-report-append-mode 50
-report-cg 50
-report-diagnostics 50
-report-file...50
-report-format 50
-report-inline 50
-report-option 50
-report-vector 51
retain 63
RETURN Statement 108
Rounding Mode 130

S

-S..32
-save 49
scalar data 67
Search 72
select_concurrent 63
select_vector... 63
-shared 53
shortloop 63
Short-loop 74
shortloop_reduction 63
Side Effects of Optimization 67
signed zero 121
sparse 64
Specifications 122
Statement Continuation............................... 109
-static 53
-static-nec 53
-std... 49
-stdlib 53
STOP Statement 108
Subscript Expression 112
Substring Expression.................................. 112
Sum or inner product 70
SX Compatibility 314
--sysroot 54

T

TMPDIR 11
-traceback 46
-traditional.. 52
Troubleshooting.. 291

U

-U 52
Unformatted Records 126
UNIX System Function Interface.................. 210
Unnamed File 130
unroll 64
unroll_complete.. 64
-use 49

V

-v 55
VE_ADVANCEOFF 12
VE_ERRCTL_ALLOCATE 13

VE_ERRCTL_DEALLOCATE............................13
VE_FMTIO_OFFLOAD..................................13
VE_FMTIO_OFFLOAD_THRESHOLD 13
VE_FORT..14
VE_FORT_ABORT 14
VE_FORT_ACCUMULATE_THREAD_CPU_TIME .14
VE_FORT_DEFAULTFILE 14
VE_FORT_EXPRCW.....................................15
VE_FORT_FILEINF.....................................16
VE_FORT_FMT_NO_WRAP_MARGIN 16
VE_FORT_FMTBUF 16
VE_FORT_FOR_PRINT 17
VE_FORT_FOR_READ 17
VE_FORT_FOR_TYPE 18
VE_FORT_MEM_BLOCKSIZE 125
VE_FORT_NML_DELIM_BLANK......................18
VE_FORT_NML_REPEAT_FORM 19
VE_FORT_NORCW................................ 18, 19
VE_FORT_PARTRCW...................................20
VE_FORT_PAUSE..21
VE_FORT_RECLUNIT 21
VE_FORT_RECORDBUF 21
VE_FORT_SETBUF......................................22
VE_FORT_SUBRCW 23
VE_FORT_UFMTADJUST 24
VE_FORT_UFMTENDIAN...............................25
VE_FORT_UFMTENDIAN_NOVEC...................26
VE_FPE_ENABLE 27
VE_INIT_HEAP..28
VE_INIT_STACK...28
VE_LD_LIBRARY_PATH 29
VE_LIBRARY_PATH.....................................11
VE_NODE_NUMBER....................................29
VE_OMP_NUM_THREADS 12
VE_OMP_STACKSIZE 12
VE_PROGINF 29
VE_TRACEBACK 29
VE_TRACEBACK_DEPTH 30
VE1/VE3 Compatibility 306
vector 64
vector data 67
vector_threshold 64
Vectorization 67
Vectorization Module 97
--version.. 55
vob 64
vovertake 64
vreg 65
vwork.. 65

W

-w... 50
-Wa... 52
-Wall 49
-Werror 49
-Wextension 49
-Wl 53
-Wobsolescent.. 49
-Woverflow 49
-Woverflow-errors....................................... 49
-Wp 52
-Wunmatched-subscript 49
-Wunmatched-subscript-errors..................... 49

X

-x 32
-Xassembler 52
-Xlinker 54

Z

-z... 54