
# NVIDIA HPC Compilers Reference Guide

***Release 25.5***

**NVIDIA Corporation**

**May 19, 2025**

# Contents

# NVIDIA HPC Compilers Reference Guide

## Preface

This guide is part of a set of manuals that describe how to use the NVIDIA HPC Fortran, C++ and C compilers. These compilers include the *NVFORTRAN*, *NVC++* and *NVC* compilers. They work in conjunction with an assembler, linker, libraries and header files on your target system, and include a CUDA toolchain, libraries and header files for GPU computing. You can use the NVIDIA HPC compilers to develop, optimize and parallelize applications for NVIDIA GPUs and x86-64 and Arm Server multicore CPUs.

The *NVIDIA HPC Compilers User's Guide* provides operating instructions for the NVIDIA HPC compilers command-level development environment. The *NVIDIA HPC Compilers Reference Guide* contains details concerning the NVIDIA compilers' interpretation of the Fortran, C++ and C language standards, implementation of language extensions, and command-level compilation. Users are expected to have previous experience with or knowledge of the Fortran, C++ and C programming languages. These guides do not teach the Fortran, C++ or C programming languages.

## Audience Description

This manual is intended for scientists and engineers using the NVIDIA HPC compilers. To use these compilers, you should be aware of the role of high-level languages, such as Fortran, C++ and C as well as parallel programming models such as CUDA, OpenACC and OpenMP in the software development process, and you should have some level of understanding of programming. The NVIDIA HPC compilers are available on a variety of NVIDIA GPUs and x86-64 and Arm CPU-based platforms and operating systems. You need to be familiar with the basic commands available on your system.

## Compatibility and Conformance to Standards

Your system needs to be running a properly installed and configured version of the NVIDIA HPC compilers. For information on installing NVIDIA HPC compilers, refer to the Release Notes and Installation Guide included with your software.

For further information, refer to the following:

- American National Standard Programming Language FORTRAN, ANSI X3. -1978 (1978).
- ISO/IEC 1539-1 : 1991, Information technology – Programming Languages – Fortran, Geneva, 1991 (Fortran 90).
- ISO/IEC 1539-1 : 1997, Information technology – Programming Languages – Fortran, Geneva, 1997 (Fortran 95).
- ISO/IEC 1539-1 : 2004, Information technology – Programming Languages – Fortran, Geneva, 2004 (Fortran 2003).
- ISO/IEC 1539-1 : 2010, Information technology – Programming Languages – Fortran, Geneva, 2010 (Fortran 2008).
- ISO/IEC 1539-1 : 2018, Information technology – Programming Languages – Fortran, Geneva, 2018 (Fortran 2018).
- Fortran 95 Handbook Complete ISO/ANSI Reference, Adams et al, The MIT Press, Cambridge, Mass, 1997.
- The Fortran 2003 Handbook, Adams et al, Springer, 2009.

- OpenACC Application Program Interface, Version 2.7, November 2018, http://www.openacc.org.
- OpenMP Application Program Interface, Version 5.0, November 2018, http://www.openmp.org.
- Programming in VAX Fortran, Version 4.0, Digital Equipment Corporation (September, 1984).
- IBM VS Fortran, IBM Corporation, Rev. GC26-4119.
- Military Standard, Fortran, DOD Supplement to American National Standard Programming Language Fortran, ANSI x.3-1978, MIL-STD-1753 (November 9, 1978).
- American National Standard Programming Language C, ANSI X3.159-1989.
- ISO/IEC 9899:1990, Information technology – Programming Languages – C, Geneva, 1990 (C90).
- ISO/IEC 9899:1999, Information technology – Programming Languages – C, Geneva, 1999 (C99).
- ISO/IEC 9899:2011, Information Technology – Programming Languages – C, Geneva, 2011 (C11).
- ISO/IEC 14882:2011, Information Technology – Programming Languages – C++, Geneva, 2011 (C++11).
- ISO/IEC 14882:2014, Information Technology – Programming Languages – C++, Geneva, 2014 (C++14).
- ISO/IEC 14882:2017, Information Technology – Programming Languages – C++, Geneva, 2017 (C++17).

### Organization

This manual contains detailed reference information about specific aspects of the compiler, such as the details of compiler options, directives, data types supported, and more. It contains these sections:

*Fortran, C++ and C Data Types* describes the data types that are supported by the NVIDIA HPC Fortran, C++ and C compilers.

*Command-Line Options Reference* provides a detailed description of most command-line options.

*C++ Name Mangling* describes the name mangling facility and explains the transformations of names of entities to names that include information on aspects of the entity's type and a fully qualified name.

*Runtime Environment* describes details related to compiler code generation, including register conventions and calling conventions for Linux/x86-64 and Linux/Arm processor-based systems.

*C++ Dialect Supported* lists more details of the version of the C++ language that NVC++ supports.

*x86-64 C++ and C MMX/SSE/AVX Intrinsics* provides tables that list the MMX and SSE/SSE2/SSE3/SSSE3/SSE4a/ABM/AVX Inline Intrinsics supported in C++ and C programs.

*Messages* provides a list of Fortran compiler error messages.

### Hardware and Software Constraints

This guide describes versions of the NVIDIA HPC compilers that target NVIDIA GPUs and x86-64 and Arm CPUs. Details concerning environment-specific values and defaults and system-specific features or limitations are presented in the release notes delivered with the NVIDIA HPC compilers.

## Conventions

This guide uses the following conventions:

***italic***
is used for emphasis.

**`Constant Width`**
is used for filenames, directories, arguments, options, examples, and for language statements in the text, including assembly language statements.

**Bold**
is used for commands.

**[ item1 ]**
in general, square brackets indicate optional items. In this case item1 is optional. In the context of p/t-sets, square brackets are required to specify a p/t-set.

**{ item2 | item 3 }**
braces indicate that a selection is required. In this case, you must select either item2 or item3.

**filename ...**
ellipsis indicate a repetition. Zero or more of the preceding item may occur. In this example, multiple filenames are allowed.

**`FORTRAN`**
Fortran language statements are shown in the text of this guide using a reduced fixed point size.

**`C++ and C`**
C++ and C language statements are shown in the test of this guide using a reduced fixed point size.

## Terms

A number of terms related to systems, processors, compilers and tools are used throughout this guide. For example:

| | | | |
|---|---|---|---|
| accelerator | FMA | -mcmodel=medium | shared library |
| AVX | host | -mcmodel=small | SIMD |
| CUDA | hyperthreading (HT) | MPI | SSE |
| device | large arrays | MPICH | static linking |
| driver | linux86-64 | NUMA | x86-64 |
| DWARF | LLVM | OpenPOWER | Arm |
| dynamic library | multicore | ppc64le | Aarch64 |

The following table lists the NVIDIA HPC compilers and their corresponding commands:

Table 1: Table 1. NVIDIA HPC Compilers and Commands

| Compiler or Tool | Language or Function | Command |
|---|---|---|
| NVFORTRAN | ISO/ANSI Fortran 2003 | nvfortran |
| NVC++ | ISO/ANSI C++17 with GNU compatibility | nvc++ |
| NVC | ISO/ANSI C11 | nvc |

In general, the designation NVFORTRAN is used to refer to the NVIDIA Fortran compiler, and nvfortran is used to refer to the command that invokes the compiler. A similar convention is used for each of the NVIDIA HPC compilers.

For simplicity, examples of command-line invocation of the compilers generally reference the **nvfortran** command, and most source code examples are written in Fortran. Use of *NVC++* and *NVC* is consistent with *NVFORTRAN*, though there are command-line options and features of these compilers that do not apply to *NVFORTRAN*, and vice versa.

There are a wide variety of x86-64 CPUs in use. Most of these CPUs are forward-compatible, but not backward-compatible, meaning that code compiled to target a given processor will not necessarily execute correctly on a previous-generation processor.

A table listing the processor options that NVIDIA HPC compilers support is available in the Release Notes. The table also includes the features utilized by the compilers that distinguish them from a compatibility standpoint.

In this manual, the convention is to use “x86-64” to specify the group of CPUs that are x86-compatible, 64-bit enabled, and run a 64-bit operating system. x86-64 processors can differ in terms of their support for various prefetch, SSE and AVX instructions. Where such distinctions are important with respect to a given compiler option or feature, it is explicitly noted in this manual.

This section describes the scalar and aggregate data types recognized by the NVIDIA Fortran, C++ and C compilers, the format and alignment of each type in memory, and the range of values each type can have on 64-bit operating systems.

# Chapter 1. Fortran Data Types

## 1.1. Fortran Scalars

A scalar data type holds a single value, such as the integer value 42 or the real value 112.6. The next table lists Fortran scalar data types, their size, format and range. *Table 3* shows the range and approximate precision for Fortran real data types. *Table 4* shows the alignment for different scalar data types. The alignments apply to all scalars, whether they are independent or contained in an array, a structure or a union.

Table 1: Table 2. Representation of Fortran Data Types

| **Fortran Data Type** | **Format** | **Range** |
|---|---|---|
| INTEGER | 2's complement integer | $-2^{31}$ to $2^{31}-1$ |
| INTEGER*2 | 2's complement integer | -32768 to 32767 |
| INTEGER*4 | 2's complement integer | $-2^{31}$ to $2^{31}-1$ |
| INTEGER*8 | 2's complement integer | $-2^{63}$ to $2^{63}-1$ |
| LOGICAL | 32-bit value | true or false |
| LOGICAL*1 | 8-bit value | true or false |
| LOGICAL*2 | 16-bit value | true or false |
| LOGICAL*4 | 32-bit value | true or false |
| LOGICAL*8 | 64-bit value | true or false |
| BYTE | 2's complement | -128 to 127 |
| REAL | Single-precision floating point | $10^{-37}$ to $10^{38(1)}$ |
| REAL*2 | Half-precision floating point (binary16) | $10^{-4}$ to $10^{5(1)}$ |
| REAL*4 | Single-precision floating point | $10^{-37}$ to $10^{38(1)}$ |
| REAL*8 | Double-precision floating point | $10^{-307}$ to $10^{308(1)}$ |
| DOUBLE PRECISION | Double-precision floating point | $10^{-307}$ to $10^{308(1)}$ |
| COMPLEX | Single-precision floating point | $10^{-37}$ to $10^{38(1)}$ |
| DOUBLE COMPLEX | Double-precision floating point | $10^{-307}$ to $10^{308(1)}$ |
| COMPLEX*16 | Double-precision floating point | $10^{-307}$ to $10^{308(1)}$ |
| CHARACTER*n | Sequence of n bytes | |

(1) Approximate value

The logical constants `.TRUE.` and `.FALSE.` are all ones and all zeroes, respectively. Internally, the value of a logical variable is true if the least significant bit is one and false otherwise. When the option `-Munixlogical` is set, a logical variable with a non-zero value is true and with a zero value is false.

**Note:** A variable of logical type may appear in an arithmetic context, and the logical type is then treated as an integer of the same size.

Table 2: Table 3. Real Data Type Ranges

| Data Type | Binary Range | Decimal Range | Digits of Precision |
|---|---|---|---|
| REAL | $-2^{-126}$ to $2^{128}$ | $10^{-37}$ to $10^{38(1)}$ | 7–8 |
| REAL*2 | $-2^{-14}$ to $2^{16}$ | $10^{-4}$ to $10^{5(1)}$ | 3–4 |
| REAL*8 | $-2^{-1022}$ to $2^{1024}$ | $10^{-307}$ to $10^{308(1)}$ | 15–16 |

Table 3: Table 4. Scalar Type Alignment

| This Type... | ...Is aligned on this size boundary |
|---|---|
| LOGICAL*1 | 1-byte |
| LOGICAL*2 | 2-byte |
| LOGICAL*4 | 4-byte |
| LOGICAL*8 | 8-byte |
| BYTE | 1-byte |
| INTEGER*2 | 2-byte |
| INTEGER*4 | 4-byte |
| INTEGER*8 | 8-byte |
| REAL*2 | 2-byte |
| REAL*4 | 4-byte |
| REAL*8 | 8-byte |
| COMPLEX*8 | 4-byte |
| COMPLEX*16 | 8-byte |

## 1.2. FORTRAN real(2)

The NVFORTRAN compiler supports `real(2)` data type which makes it possible to declare and use data in half precision floating point. It is explicitly required to use the `kind` attribute with value of 2 on `real` data type to take advantage of this support. The following operators are supported for this data type: `+ , -, *, /, .lt., .le., .gt., .ge., .eq.,.ne.`.

There are several ways to create `real(2)` constants:

```
! Using kind attribute of 2 by appending _2 to the floating point value:
real(2) :: val1 = 2.0_2
! Using a hexadecimal constant:
real(2) :: val2 = z'4000'
! Explicitly calling real() intrinsic with the value to be converted:
real(2) :: val3 = real(2, kind=2)
! Implicitly relying on compiler to convert value to real(2):
real(2) :: val4 = 2d0
```

Half precision native support is not available on all of the architecture targets that NVFORTRAN supports. It is still possible to use this type, but be aware that implementation relies on conversion to `real(4)`, handling operation in `real(4)`, and then converting back to `real(2)`. NVIDIA GPUs which support CUDA Compute Capability 6.0 and above implement operations natively and do not rely on conversion.

Half precision is represented as IEEE 754 binary16. Out of the 16-bits available to represent the floating point value, one bit is used for sign, five bits are used for exponent, and ten bits are used for significand. When encountering values that cannot be precisely represented in the format, such as when adding two `real(2)` numbers, IEEE 754 defines rounding rules. In the case of `real(2)`, the default rule is round-to-nearest with ties-to-even property which is described in detail in the IEEE 754-2008 standard in section 4.3.1. This format has a small dynamic range and thus values greater than 65520 are rounded to infinity.

## 1.3. FORTRAN 77 Aggregate Data Type Extensions

The NVFORTRAN compiler supports de facto standard extensions to FORTRAN 77 that allow for aggregate data types. An aggregate data type consists of one or more scalar data type objects. You can declare the following aggregate data types:

- An *array* consists of one or more elements of a single data type placed in contiguous locations from first to last.
- A *structure* can contain different data types. The members are allocated in the order they appear in the definition but may not occupy contiguous locations.
- A *union* is a single location that can contain any of a specified set of scalar or aggregate data types. A union can have only one value at a time. The data type of the union member to which data is assigned determines the data type of the union after that assignment.

The alignment of an array, a structure or union (an aggregate) affects how much space the object occupies and how efficiently the processor can address members. Arrays use the alignment of their members.

**Array types**
align according to the alignment of the array elements. For example, an array of INTEGER*2 data aligns on a 2-byte boundary. The exception to this rule is that aligment of REAL*2 arrays is on a 4-byte boundary.

**Structures and Unions**
align according to the alignment of the most restricted data type of the structure or union. In the next example, the union aligns on a 4-byte boundary since the alignment of c, the most restrictive element, is four.

```
STRUCTURE /astr/
UNION
 MAP
 INTEGER*2 a ! 2 bytes
 END MAP
 MAP
 BYTE b ! 1 byte
 END MAP
 MAP
 INTEGER*4 c ! 4 bytes
 END MAP
END UNION
END STRUCTURE
```

Structure alignment can result in unused space called *padding*. Padding between members of the structure is called *internal padding*. Padding between the last member and the end of the space is called *tail padding*.

The offset of a structure member from the beginning of the structure is a multiple of the member's alignment. For example, since an INTEGER*2 aligns on a 2-byte boundary, the offset of an INTEGER*2member from the beginning of a structure is a multiple of two bytes.

## 1.4. Fortran 90 Aggregate Data Types (Derived Types)

The Fortran 90 standard added formal support for aggregate data types. The TYPE statement begins a derived type data specification or declares variables of a specified user-defined type. For example, the following would define a derived type ATTENDEE:

```
TYPE ATTENDEE
 CHARACTER(LEN=30) NAME
 CHARACTER(LEN=30) ORGANIZATION
 CHARACTER (LEN=30) EMAIL
END TYPE ATTENDEE
```

In order to declare a variable of type ATTENDEE and access the contents of such a variable, code such as the following would be used:

```
TYPE (ATTENDEE) ATTLIST(100)
. . .
ATTLIST(1)%NAME = 'JOHN DOE'
```

# Chapter 2. C and C++ Data Types

## 2.1. C and C++ Scalars

*Table 5* lists C and C++ scalar data types, providing their size and format. The alignment of a scalar data type is equal to its size. *Table 6* shows scalar alignments that apply to individual scalars and to scalars that are elements of an array or members of a structure or union. Wide characters are supported (character constants prefixed with an L). The size of each wide character is 4 bytes.

Table 1: Table 5. C/C++ Scalar Data Types

| Data Type | Size (bytes) | Format | Range |
|---|---|---|---|
| unsigned char | 1 | ordinal | 0 to 255 |
| signed char | 1 | 2's complement integer | -128 to 127 |
| char | 1 | 2's complement integer | -128 to 127 |
| char | 1 | ordinal | 0 to 255 |
| unsigned short | 2 | ordinal | 0 to 65535 |
| [signed] short | 2 | 2's complement integer | -32768 to 32767 |
| unsigned int | 4 | ordinal | 0 to $2^{32}$ -1 |
| [signed] int | 4 | 2's complement integer | $-2^{31}$ to $2^{31}$-1 |
| [signed] long [int] (win64) | 4 | 2's complement integer | $-2^{31}$ to $2^{31}$-1 |
| [signed] long [int] (linux86-64) | 8 | 2's complement integer | $-2^{63}$ to $2^{63}$-1 |
| unsigned long [int] (win64) | 4 | ordinal | 0 to $2^{32}$-1 |
| unsigned long [int] (linux86-64) | 8 | ordinal | 0 to $2^{64}$-1 |
| [signed] long long [int] | 8 | 2's complement integer | $-2^{63}$ to $2^{63}$-1 |
| unsigned long long [int] | 8 | ordinal | 0 to $2^{64}$-1 |
| [signed] __int128 | 16 | 2's complement integer | $-2^{127}$ to $2^{127}$-1 |
| unsigned __int128 | 16 | ordinal | 0 to $2^{128}$-1 |
| float | 4 | IEEE single-precision floating-point | $10^{-37}$ to $10^{38(1)}$ |
| double | 8 | IEEE double-precision floating-point | $10^{-307}$ to $10^{308(1)}$ |
| long double | 16 | IEEE extended-precision floating-point | $10^{-4931}$ to $10^{4932(1)}$ |
| long double | 16 | IBM double-double | $10^{-307}$ to $10^{308(1)}$ |
| bit field(2) (unsigned value) | 1 to 32 bits | ordinal | 0 to $2^{size}$-1, where size is the number of bits in the bit field |
| bit field(2) (signed value) | 1 to 32 bits | 2's complement integer | $-2^{size-1}$ to $2^{size-1}$-1, where size is the number of bits in the bit field |
| pointer (32-bit operating system) | 4 | address | 0 to $2^{32}$-1 |
| pointer | 8 | address | 0 to $2^{64}$-1 |
| enum | 4 | 2's complement integer | $-2^{31}$ to $2^{31}$-1 |

(1) Approximate value

(2) Bit fields occupy as many bits as you assign them, up to 4 bytes, and their length need not be a multiple of 8 bits (1 byte)

Table 2: Table 6. Scalar Alignment

| Data Type | Alignment on this size boundary |
|---|---|
| char | 1-byte boundary, signed or unsigned. |
| short | 2-byte boundary, signed or unsigned. |
| int | 4-byte boundary, signed or unsigned. |
| enum | 4-byte boundary. |
| pointer | 8-byte boundary. |
| float | 4-byte boundary. |
| double | 8-byte boundary. |
| long double | 8-byte boundary. |
| long double (64-bit operating system) | 16-byte boundary. |
| long [int] linux86-64 | 8-byte boundary, signed or unsigned. |
| long long [int] | 8-byte boundary, signed or unsigned. |

## 2.2. C and C++ Aggregate Data Types

An aggregate data type consists of one or more scalar data type objects. You can declare the following aggregate data types:

**array**
consists of one or more elements of a single data type placed in contiguous locations from first to last.

**class**
(C++ only) is a class that defines an object and its member functions. The object can contain fundamental data types or other aggregates including other classes. The class members are allocated in the order they appear in the definition but may not occupy contiguous locations.

**struct**
is a structure that can contain different data types. The members are allocated in the order they appear in the definition but may not occupy contiguous locations. When a struct is defined with member functions, its alignment rules are the same as those for a class.

**union**
is a single location that can contain any of a specified set of scalar or aggregate data types. A union can have only one value at a time. The data type of the union member to which data is assigned determines the data type of the union after that assignment.

## 2.3. Class and Object Data Layout

Class and structure objects with no virtual entities and with no base classes, that is just direct data field members, are laid out in the same manner as C structures. The following section describes the alignment and size of these C-like structures. C++ classes (and structures as a special case of a class) are more difficult to describe. Their alignment and size is determined by compiler generated fields in addition to user-specified fields. The following paragraphs describe how storage is laid out for more general classes. The user is warned that the alignment and size of a class (or structure) is dependent on the existence and placement of direct and virtual base classes and of virtual function information. The information that follows is for informational purposes only, reflects the current implementation, and is subject to change. Do not make assumptions about the layout of complex classes or structures.

All classes are laid out in the same general way, using the following pattern (in the sequence indicated):

- First, storage for all of the direct base classes (which implicitly includes storage for non-virtual indirect base classes as well):
  - When the direct base class is also virtual, only enough space is set aside for a pointer to the actual storage, which appears later.
  - In the case of a non-virtual direct base class, enough storage is set aside for its own non-virtual base classes, its virtual base class pointers, its own fields, and its virtual function information, but no space is allocated for its virtual base classes.
- Next, storage for the class's own fields.
- Next, storage for virtual function information (typically, a pointer to a virtual function table).
- Finally, storage for its virtual base classes, with space enough in each case for its own non-virtual base classes, virtual base class pointers, fields, and virtual function information.

## 2.4. Aggregate Alignment

The alignment of an array, a structure or union (an aggregate) affects how much space the object occupies and how efficiently the processor can address members.

**Arrays**
align according to the alignment of the array elements. For example, an array of short data type aligns on a 2-byte boundary.

**Structures and Unions**
align according to the most restrictive alignment of the enclosing members. In the following example, the union un1 aligns on a 4-byte boundary since the alignment of c, the most restrictive element, is four:

```
union un1 {
 short a; /* 2 bytes */
 char b; /* 1 byte */
 int c; /* 4 bytes */
 };
```

Structure alignment can result in unused space, called padding. Padding between members of a structure is called internal padding. Padding between the last member and the end of the space occupied by the structure is called tail padding. *Figure 1* illustrates structure alignment. Consider the following structure:

```
struct strc1 {
 char a; /* occupies byte 0 */
 short b; /* occupies bytes 2 and 3 */
 char c; /* occupies byte 4 */
 int d; /* occupies bytes 8 through 11 */
 };
```

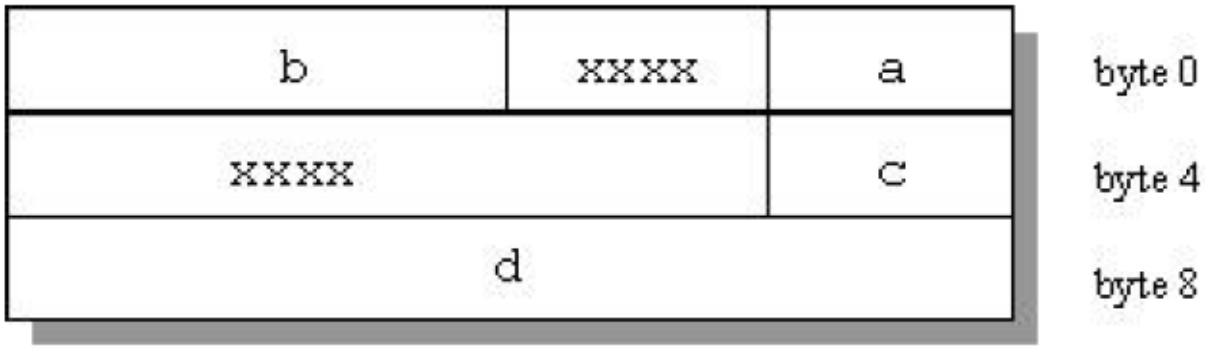


Fig. 1: Figure 1. Internal Padding in a Structure

*Figure 2* shows how tail padding is applied to a structure aligned on a doubleword (8 byte) boundary.

```
struct strc2{
 int m1[4]; /* occupies bytes
0 through 15 */
 double m2; /* occupies bytes 16 through 23 */
 short m3; /* occupies bytes 24 and 25 */
} st;
```

## 2.5. Bit-field Alignment

Bit-fields have the same size and alignment rules as other aggregates, with several additions to these rules:

- Bit-fields are allocated from right to left.
- A bit-field must entirely reside in a storage unit appropriate for its type. Bit-fields never cross unit boundaries.
- Bit-fields may share a storage unit with other structure/union members, including members that are not bit-fields.
- Unnamed bit-field's types do not affect the alignment of a structure or union.

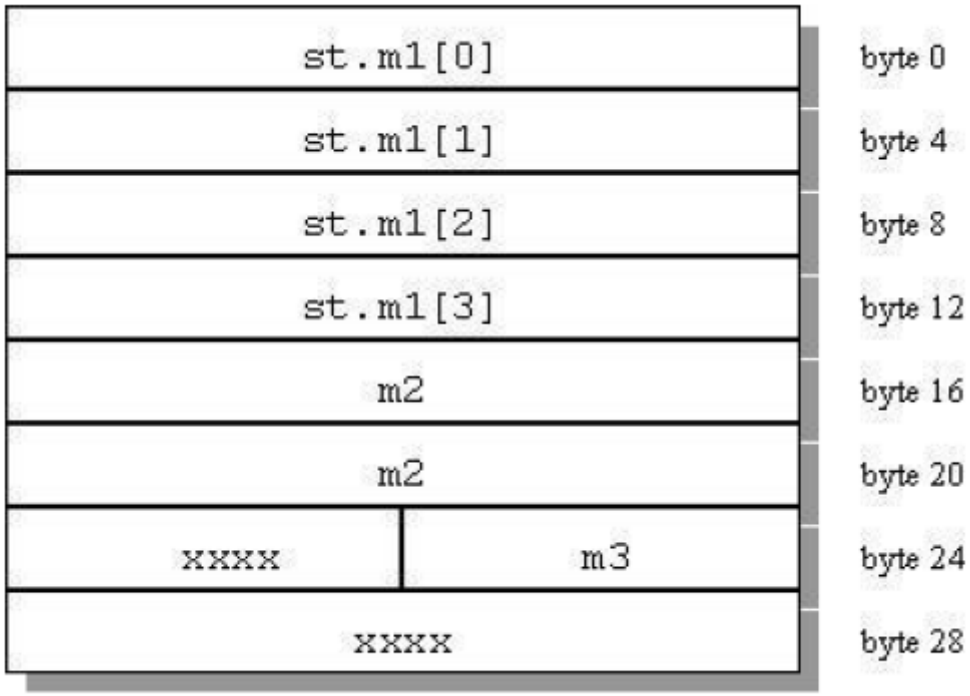


Fig. 2: Figure 2. Tail Padding in a Structure

## 2.6. Other Type Keywords in C and C++

The void data type is neither a scalar nor an aggregate. You can use void or void* as the return type of a function to indicate the function does not return a value, or as a pointer to an unspecified data type, respectively.

The const and volatile type qualifiers do not in themselves define data types, but associate attributes with other types. Use const to specify that an identifier is a constant and is not to be changed. Use volatile to prevent optimization problems with data that can be changed from outside the program, such as memory-mapped I/O buffers.

Extended integer types __int128 and unsigned __int128 are now supported by NVC and NVC++. 128-bit integer support can be turned on with the -Mint128 flag. Note, 128-bit integer support is not supported with OpenMP, OpenACC and CUDA.

# Chapter 3. Command-Line Options Reference

A command-line option allows you to specify specific behavior when a program is compiled and linked. Compiler options perform a variety of functions, such as setting compiler characteristics, describing the object code to be produced, controlling the diagnostic messages emitted, and performing some preprocessor functions. Most options that are not explicitly set take the default settings. This reference section describes the syntax and operation of each compiler option. For easy reference, the options are arranged in alphabetical order.

For an overview and tips on options usage and which options are best for which tasks, refer to the 'Using Command-line Options' section of the HPC Compilers User Guide, which also provides summary tables of the different options.

This section uses the following notation:

**[item]**
Square brackets indicate that the enclosed item is optional.

**{item | item}**
Braces indicate that you must select one and only one of the enclosed items. A vertical bar (|) separates the choices.

**...** Horizontal ellipses indicate that zero or more instances of the preceding item are valid.

## 3.1. HPC Compilers Option Summary

The following tables include all the HPC compiler options that are not language-specific. The options are separated by category for easier reference.

For a complete description of each option, refer to the detailed information later in this section.

### 3.1.1. Acceleration and Optimization-Related Compiler Options

The options included in the following table pertain to optimizing your program or application code.

Table 1: Table 7. Acceleration and Optimization-Related HPC Compiler Options

| Option | Description |
|---|---|
| `-acc[={gpu\|multicore}]` | Enable OpenACC directives for GPUs (default) or multicore CPUs. |
| `-cuda[={charstring\|madconst}]` | Enable CUDA Fortran features for GPUs (default for .cuf/.CUF files). |
| `-fast` | Enable a generally optimal set of CPU code generation flags including SIMD vectorization. |
| `-gpu=[...]` | Specify details of GPU code generation including compute capability, CUDA version and more. |
| `-M<nvflag>` | Selects variations for code generation and optimization. |
| `-mp[={gpu\|multicore}[,[no]align]][,[no]autopar]]` | Enable OpenMP code generation for GPUs and multicore CPUs; implies -Mrecursive in Fortran |
| `-O<level>` | Specifies code optimization level where <level> is 0, 1, 2, 3, 4 or fast. |
| `-stdpar[={gpu\|multicore}]` | Enable parallelization/offload of Standard C++ and Fortran parallel constructs; default is -stdpar=gpu. |
| `-target={gpu\|multicore}` | Specify code-generation target for -acc, -mp, -stdpar. |

### 3.1.2. Build-Related Options

The options included in the following table pertain to the initial building of your program or application.

Table 2: Table 8. Build-Related Compiler Options

| Option | Description |
|---|---|
| `-#` | Display invocation information. |
| `-Bdynamic` | Compiles for and links to the shared object version of the NVIDIA runtime libraries. |
| `-static` | Passed to the linker to specify static binding. |
| `-static-nvidia` | Statically link in the NVIDIA runtime libraries, while using dynamic linking for the system libraries; implies -Mnorpath. |
| `-c` | Stops after the assembly phase and saves the object code in `filename.o`. |

continues on next page

Table 2 – continued from previous page

| Option | Description |
|---|---|
| -c++libs | Append GNU-compatible C++ libraries to the link line. |
| -D<args> | Defines a preprocessor macro. |
| -dryrun | Shows but does not execute driver commands. |
| -drystdinc | Displays the standard include directories and then exits the compiler. |
| -E | Stops after the preprocessing phase and displays the preprocessed file on the standard output. |
| -F | Stops after the preprocessing phase and saves the preprocessed file in filename.f. This option is only valid for the NVIDIA Fortran compilers. |
| --flagcheck | Simply return zero status if flags are correct. |
| -flags | Display valid driver options. |
| -fmax-errors=<n> | Set compiler error limit to <n> |
| -fortranlibs | Append NVFORTRAN runtime libraries to the link line. |
| -fpic | Generate position-independent code. |
| -fPIC | Equivalent to -fpic. |
| --gcc-toolchain=<path> | Specify gcc toolchain location using path to gcc directory or gcc executable. |
| -help, --help | Display driver help message. |
| -I<dirname> | Adds a directory to the search path for #include files. |
| -i2 | Treat INTEGER variables as 2 bytes. |
| -i4 | Treat INTEGER variables as 4 bytes. |
| -i8 | Treat INTEGER and LOGICAL variables as 8 bytes and use 64-bits for INTEGER*8 operations. |
| -K<flag> | Requests special compilation semantics with regard to conformance to IEEE 754. |
| -L<dirname> | Specifies a directory to search for libraries. |
| -l<library> | Loads a library. |
| -m | Displays a link map on the standard output. |
| -M<pgflag> | Selects variations for code generation and optimization. |
| -mcmodel=medium | Generate code which supports the medium memory model in the Linux environment. |
| -module <moduledir> | (Fortran only) Save/search for module files in directory <moduledir>. |
| -noswitcherror | Ignore unknown command line switches after printing an warning message. |
| -o | Names the object file. |
| --pedantic | (C++ only) Prints warnings from included <system header files> |

continues on next page

Table 2 – continued from previous page

| Option | Description |
|---|---|
| -pg or -qp | Instrument the generated executable to produce a gprof-style gmon.out sample-based profiling trace file; -qp is equivalent to -pg. |
| -R<directory> | Passed to the Linker. Hard code <directory> into the search path for shared object files. |
| -r | Creates a relocatable object file. |
| -r4 | Interpret DOUBLE PRECISION variables as REAL. |
| -r8 | Interpret REAL variables as DOUBLE PRECISION. |
| -rc file | Specifies the name of the driver's startup file. |
| -s | Strips the symbol-table information from the object file. |
| -S | Stops after the compiling phase and saves the assembly-language code in filename.s. |
| -shared | Passed to the linker. Instructs the linker to generate a shared object file. Implies -fpic. |
| -show | Display driver's configuration parameters after startup. |
| -silent | Do not print warning messages. |
| -soname | Pass the soname option and its argument to the linker. |
| -time | Print execution times for the various compilation steps. |
| -tp <target> | Specify the type of the CPU target processor. Cross-compilation is only supported within CPU Architecture families. |
| -u<symbol> | Initializes the symbol table with <symbol>, which is undefined for the linker. An undefined symbol triggers loading of the first member of an archive library. |
| -U<symbol> | Undefine a preprocessor macro. |
| -V[release_number] | Displays the version messages and other information, or allows invocation of a version of the compiler other than the default. |
| -v | Displays the compiler, assembler, and linker phase invocations. |
| -W | Passes arguments to a specific phase. |
| -Werror | Turn all warning messages into errors. |
| -w | Do not print warning messages. |
| -Xlinker <option> | Passes options to the linker. |

### 3.1.3. Debug-Related Compiler Options

The options included in the following table pertain to debugging your program or application.

Table 3: Table 9. Debug-Related Compiler Options

| Option | Description |
|---|---|
| `-C` | (Fortran only) Generates code to check array bounds. |
| `-E` | Stops after the preprocessing phase and displays the preprocessed file on the standard output. |
| `--flagcheck` | Simply return zero status if flags are correct. |
| `-flags` | Display valid driver options. |
| `-g` | Includes debugging information in the object module; sets the optimization level to zero unless a -O option is present on the command line. |
| `-gopt` | Includes debugging information in the object module, but forces assembly code generation identical to that obtained when `-gopt` is not present on the command line. |
| `-K<flag>` | Requests special compilation semantics with regard to conformance to IEEE 754. |
| `--keeplnk` | If the compiler generates a temporary indirect file for a long linker command, preserves the temporary file instead of deleting it. |
| `-M<pgflag>` | Selects variations for code generation and optimization. |
| `-[no]traceback` | Adds debug information for runtime traceback for use with the environment variable `NVCOMPILER_TERM`. |

### 3.1.4. Linking and Runtime-Related Compiler Options

The options included in the following table pertain to defining parameters related to linking and running your program or application.

Table 4: Table 10. Linking and Runtime-Related Compiler Options

| Option | Description |
|---|---|
| -Bdynamic | Compiles for and links to the shared object version of the NVIDIA runtime libraries. |
| -static | Passed to the linker to specify static binding. |
| -static-nvidia | Statically link in the NVIDIA runtime libraries, while using dynamic linking for the system libraries; implies -Mnorpath. |
| -byteswapio | (Fortran only) Swap bytes from big-endian to little-endian or vice versa on input/output of unformatted data. |
| -c++libs | Append GNU-compatible C++ runtime libraries to the link line. |
| -fortranlibs | Append NVFORTRAN runtime libraries to the link line. |
| -fpic or -fPIC | Generate position-independent code. |
| -g77libs | Allow object files generated by gfortran or g77 to be linked into NVIDIA main programs. |
| -i2 | Treat INTEGER variables as 2 bytes. |
| -i4 | Treat INTEGER variables as 4 bytes. |
| -i8 | Treat INTEGER and LOGICAL variables as 8 bytes and use 64-bits for INTEGER*8 operations. |
| -K<flag> | Requests special compilation semantics with regard to conformance to IEEE 754. |
| -M<pgflag> | Selects variations for code generation and optimization. |
| -mcmodel=medium | Generate code which supports the medium memory model in 64-bit Linux environments. |
| -[no]nvmalloc | Link in a custom host memory allocator library. |
| -shared | Passed to the linker. Instructs the linker to generate a shared object file. Implies -fpic. |
| -soname | Pass the soname option and its argument to the linker. |
| -tp <target> | Specify the type of the CPU target processor. Cross-compilation is only supported within CPU Architecture families. |
| -Xlinker <option> | Pass options to the linker. |

# 3.2. Generic Compiler Options

The following descriptions are for compiler options common to the NVIDIA HPC Fortran, C++ and C compilers. For easy reference, the options are arranged in alphabetical order. For a list of options by tasks, refer to the tables in the beginning of this section.

## 3.2.1. -#

Displays the invocations of the compiler, assembler and linker.

**Default**

The compiler does not display individual phase invocations.

**Usage**

The following command-line requests verbose invocation information.

```
$ nvfortran -# prog.f
```

**Description**

The -# option displays the invocations of the compiler, assembler and linker. These invocations are command-lines created by the driver from your command-line input and the default value.

**Related options**

*-Minfo[=option [,option,...]]*, *-V[release_number]*, *-v*

## 3.2.2. -[no]acc

Enable [disable] OpenACC directives. The following suboptions may be used following an equals sign ("="), with multiple sub-options separated by commas:

**gpu** OpenACC directives are compiled for GPU execution only.

**host**
Compile for serial execution on the host CPU.

**multicore**
Compile for parallel execution on the host CPU.

**legacy**
Suppress warnings about deprecated NVIDIA accelerator directives.

**[no]autopar**
Enable [disable] loop autoparallelization within acc parallel. The default is to autoparallelize, that is, to enable loop autoparallelization.

**[no]routineseq**
Compile every routine for the devicee. The default behavior is to not treat every routine as a seq directive.

**strict**
Instructs the compiler to issue warnings for non-OpenACC accelerator directives.

**sync**
Ignore async clauses

**verystrict**
Instructs the compiler to fail with an error for any non-OpenACC accelerator directive.

**[no]wait**
Wait for each device kernel to finish. Kernel launching is blocked by default unless the async clause is used.

**Default**

By default OpenACC directives are compiled for GPU and sequential CPU host execution (i.e. equivalent to explicitly setting `-acc=gpu,host`).

**Usage**

The following command-line requests that OpenACC directives be enabled and that an error be issued for any non-OpenACC accelerator directive.

```
$ nvfortran -acc=verystrict prog.f
```

**Predefined Macros**

The following macros corresponding to the target compiled for are added implicitly:

- `__NVCOMPILER_OPENACC_GPU` when the OpenACC directives are compiled for GPU.
- `__NVCOMPILER_OPENACC_MULTICORE` when the OpenACC directives are compiled for multicore CPU.
- `__NVCOMPILER_OPENACC_HOST` when the OpenACC directives are compiled for serial execution on CPU.

## 3.2.3. -Bdynamic

Compiles for and links to the shared object version of the NVIDIA HPC Compilers runtime libraries.

**Default**

Dynamic linking is the default behavior for Linux.

**Usage**

```
% nvfortran -Bdynamic myprogram.f
```

When you use the NVIDIA HPC compiler flag `-Bdynamic` to create an executable that links to the shared object form of the runtime, the executable built is smaller than one built without `-Bdynamic`. The NVIDIA HPC Compilers runtime shared object(s), however, must be available on the system where the executable is run. The `-Bdynamic` flag must be used when an executable is linked against a shared object built by the NVIDIA HPC compilers.

**Related options**

*-static*

## 3.2.4. -byteswapio

Swaps the byte-order of data in unformatted Fortran data files on input/output.

**Default**

The compiler does not byte-swap data on input/output.

**Usage**

The following command-line requests that byte-swapping be performed on input/output.

```
$ nvfortran -byteswapio myprog.f
```

**Description**

Use the `-byteswapio` option to swap the byte-order of data in unformatted Fortran data files on input/output. When this option is used, the order of bytes is swapped in both the data and record control words; the latter occurs in unformatted sequential files.

You can use this option to convert big-endian format data files produced by most legacy RISC workstations to the little-endian format used on modern Linux systems on the fly during file reads/writes.

This option assumes that the record layouts of unformatted sequential access and direct access files are the same on the systems. It further assumes that the IEEE representation is used for floating-point numbers. In particular, the format of unformatted data files produced by NVIDIA HPC Fortran compilers is identical to the format used on Sun and SGI workstations; this format allows you to read and write unformatted Fortran data files produced on those platforms from a program compiled for modern Linux platform using the `-byteswapio` option.

**Related options**

None.

## 3.2.5. -C

(Fortran only) Generates code to check array bounds.

**Default**

The compiler does not enable array bounds checking.

**Usage**

In this example, the compiler instruments the executable produced from `myprog.f` to perform array bounds checking at runtime:

```
$ nvfortran -C myprog.f
```

**Description**

Use this option to enable array bounds checking. If an array is an assumed size array, the bounds checking only applies to the lower bound. If an array bounds violation occurs during execution, an error message describing the error is printed and the program terminates. The text of the error message includes the name of the array, the location where the error occurred (the source file and the line number in the source), and information about the out of bounds subscript (its value, its lower and upper bounds, and its dimension).

**Related options**

*-M[no]bounds*

## 3.2.6. -c

Halts the compilation process after the assembling phase and writes the object code to a file.

**Default**

The compiler produces an executable file and does not use the -c option.

**Usage**

In this example, the compiler produces the object file `myprog.o` in the current directory.

```
$ nvfortran -c myprog.f
```

**Description**

Use the -c option to halt the compilation process after the assembling phase and write the object code to a file. If the input file is `filename.f`, the output file is `filename.o`.

**Related options**

*-E*, *-Mkeepasm*, *-o*, *-S*

## 3.2.7. -c++libs

Instructs the compiler to append C++ runtime libraries to the link line for programs built using NVFORTRAN.

**Default**

The NVFORTRAN compiler does not append the C++ runtime libraries to the link line.

**Usage**

In the following example the C++ runtime libraries are linked with an object file compiled with NVFORTRAN

```
$ nvfortran main.f90 mycpp.o -c++libs
```

**Description**

Use this option to instruct the NVIDIA Fortran compiler to append C++ runtime libraries to the link line.

**Related options**

*-fortranlibs*

## 3.2.8. -cuda

Enable CUDA; please refer to `-gpu` for target-specific options. The following suboptions may be used following an equals sign ("="), with multiple sub-options separated by commas:

**`charstring`**
Enable limited support for character strings in GPU kernels.

**`madconst`**
Put Module Array Descriptors in CUDA Constant Memory

**Usage**

The following command-line requests that CUDA interoperability be enabled and CUDA Fortran syntax be recognized and processed in all Fortran files.

```
$ nvfortran -cuda myprog.f
```

## 3.2.9. -cudalib

Add CUDA-optimized libraries to the link line. When no sub-option is specified the compiler will link all necessary CUDA-optimized libraries. -cudalib will use the version of the library appropriate to the CUDA version being used. The following libraries may be specified following an equals sign ("="), with multiple libraries separated by commas:

**`cublas`**
Link in the cuBLAS library.

**`cufft`**
Link in the cuFFT library.

**`cufftw`**
Link in the cuFFTW library.

**`curand`**
Link in the cuRAND library.

**`cusolver`**
Link in the cuSOLVER library.

**`cusparse`**
Link in the cuSPARSE library.

**`cutensor`**
Link in the cuTENSOR library.

**`nvblas`**
Link in the NVBLAS library.

**`nccl`**
Link in the NCCL library.

**`nvlamath`**
Link in the NVLAmath library.

**`nvshmem`**
Link in the NVSHMEM library.

### Usage

The following command-line links in all necessary CUDA libraries.

```
$ nvfortran -acc -cudalib myprog.cpp
```

## 3.2.10. -D

Creates a preprocessor macro with a given value.

**Note:** You can use the -D option more than once on a compiler command line. The number of active macro definitions is limited only by available memory.

### Syntax

```
-Dname[=value]
```

Where name is the symbolic name and value is either an integer value or a character string.

### Default

If you define a macro name without specifying a value, the preprocessor assigns the string 1 to the macro name.

### Usage

In the following example, the macro PATHLENGTH has the value 256 until a subsequent compilation. If the -D option is not used, PATHLENGTH is set to 128.

```
$ nvfortran -DPATHLENGTH=256 myprog.F
```

The source text in `myprog.F` is this:

```
 #ifndef PATHLENGTH
#define PATHLENGTH 128
#endif  SUBROUTINE SUB  CHARACTER*PATHLENGTH path
        ...
END
```

### Description

Use the -D option to create a preprocessor macro with a given value. The value must be either an integer or a character string.

You can use macros with conditional compilation to select source text during preprocessing. A macro defined in the compiler invocation remains in effect for each module on the command line, unless you remove the macro with an #undef preprocessor directive or with the -U option. The compiler processes all of the -U options in a command line after processing the -D options.

### Related options

*-U*

## 3.2.11. -d<arg>

Prints additional information from the preprocessor. [Valid only for the C compiler (nvc) ]

**Default**

No additional information is printed from the preprocessor.

**Syntax**

```
-d[D|I|M|N]
```

**-dD** Print macros and values from source files.

**-dI** Print include file names.

**-dM** Print macros and values, including predefined and command-line macros.

**-dN** Print macro names from source files.

**Usage**

In the following example, the compiler prints macro names from the source file.

```
$ nvc -dN myprog.f
```

**Description**

Use the -d<arg> option to print additional information from the preprocessor.

**Related options**

*-E*, *-D*, *-U*

## 3.2.12. -dryrun

Displays the invocations of the compiler, assembler, and linker but does not execute them.

**Default**

The compiler does not display individual phase invocations.

**Usage**

The following command line requests verbose invocation information.

```
$ nvfortran -dryrun myprog.f
```

**Description**

Use the `-dryrun` option to display the invocations of the compiler, assembler, and linker but not have them executed. These invocations are command lines created by the compiler driver from the `rc` files and the command-line supplied with `-dryrun`.

**Related options**

*-Minfo[=option [,option,...]]*, *-V[release_number]*

## 3.2.13. -drystdinc

Displays the standard include directories and then exits the compiler.

**Default**

The compiler does not display standard include directories.

**Usage**

The following command line requests a display for the standard include directories.

```
$ nvc -drystdinc myprog.c
```

**Description**

Use the `-drystdinc` option to display the standard include directories and then exit the compiler.

**Related options**

None.

## 3.2.14. -E

Halts the compilation process after the preprocessing phase and displays the preprocessed output on the standard output.

**Default**

The compiler produces an executable file.

**Usage**

In the following example the compiler displays the preprocessed `myprog.f` on the standard output.

```
$ nvc -E myprog.c
```

**Description**

Use the -E option to halt the compilation process after the preprocessing phase and display the preprocessed output on the standard output.

**Related options**

*-C*, *-c*, *-Mkeepasm*, *-o*, *-F*, *-S*

## 3.2.15. -F

Stops compilation after the preprocessing phase.

**Default**

The compiler produces an executable file.

**Usage**

In the following example the compiler produces the preprocessed file `myprog.f` in the current directory.

```
$ nvfortran -F myprog.F
```

**Description**

Use the -F option to halt the compilation process after preprocessing and write the preprocessed output to a file. If the input file is `filename.F`, then the output file is `filename.f`.

**Related options**

*-c*, *-E*, *-Mkeepasm*, *-o*, *-S*

## 3.2.16. -fast

Enables vectorization with SIMD instructions, cache alignment, and flushz for 64-bit targets.

**Default**

The compiler does not enable vectorization with SIMD instructions, cache alignment, and flushz.

**Usage**

In the following example the compiler produces vector SIMD code when targeting a 64-bit machine.

```
$ nvfortran -fast vadd.f95
```

**Description**

When you use this option, a generally optimal set of options is chosen for targets that support SIMD capability. In addition, the appropriate `-tp` option is automatically included to enable generation of code optimized for the type of system on which compilation is performed. This option enables vectorization with SIMD instructions, cache alignment, and flushz.

---

**Note:** Auto-selection of the appropriate `-tp` option means that programs built using the `-fast` option on a given system are not necessarily backward-compatible with older systems.

---

---

**Note:** C/C++ compilers enable `-Mautoinline` with `-fast`.

---

**Related options**

*-O<level>*, *-Munroll[=option [,option...]]*, *-Mnoframe*, *-M[no]vect[=option [,option,...]]*, *-Mcache_align*, *-tp <target>*, *-M[no]autoinline[=option[,option,...]]*

## 3.2.17. -fcx-limited-range

`-fcx-limited-range` specifies that complex division does not need range reduction. `-fno-cx-limited-range` specifies that complex division does need range reduction.

**Default**

`-fcx-limited-range` with `-Ofast`

Otherwise: `-fno-cx-limited-range`

**Related options**

*-O<level>*

## 3.2.18. –flagcheck

Causes the compiler to check that flags are correct and then exit without any compilation occuring.

**Default**

The compiler begins a compile without the additional step to first validate that flags are correct.

**Usage**

In the following example the compiler checks that flags are correct, and then exits.

```
$ nvfortran --flagcheck myprog.f
```

**Description**

Use this option to make the compiler check that flags are correct and then exit. If flags are all correct then the compiler returns a zero status. No compilation occurs.

**Related options**

None.

## 3.2.19. -fortranlibs

Instructs the C++ or C compiler to append NVFORTRAN runtime libraries to the link line.

**Default**

The C++ and compilers do not append the NVFORTRAN runtime libraries to the link line.

**Usage**

In the following example a `.c` main program is linked with an object file compiled with nvfortran.

```
$ nvc main.c myfort.o -fortranlibs
```

**Description**

Use this option to instruct the C++ or C compiler to append NVFORTRAN runtime libraries to the link line.

**Related options**

*-c++libs*

## 3.2.20. -fmax-errors=<n>

Set

**Default**

Abort compilation after a user defined error limit.

**Usage**

In the following example the compiler error limit is set to 5.

```
$ nvfortran -fmax-errors=5 myprog.f
```

Use the `-fmax-errors` option to increase or decrease the default compilation error limit.

**Related options**

*-Werror*

## 3.2.21. -fpic

Generates position-independent code suitable for inclusion in shared object (dynamically linked library) files.

**Default**

The compiler does not generate position-independent code.

**Usage**

In the following example the resulting object file, `myprog.o`, can be used to generate a shared object.

```
$ nvfortran -fpic myprog.f
```

Use the -fpic option to generate position-independent code suitable for inclusion in shared object (dynamically linked library) files.

**Related options**

*-shared*, *-fPIC*, *-R<directory>*

## 3.2.22. -fPIC

Equivalent to `-fpic`. Provided for compatibility with other compilers.

## 3.2.23. -g

Instructs the compiler to include symbolic debugging information in the object module; sets the optimization level to zero unless a -O option is present on the command line.

**Default**

The compiler does not put debugging information into the object module.

**Usage**

In the following example, the object file `myprog.o` contains symbolic debugging information.

```
$ nvfortran -c -g myprog.f
```

**Description**

Use the -g option to instruct the compiler to include symbolic debugging information in the object module. Debuggers require symbolic debugging information in the object module to display and manipulate program variables and source code.

If you specify the `-g` option on the command-line, the compiler sets the optimization level to -O0 (zero), unless you specify the `-O` option. For more information on the interaction between the `-g` and `-O` options, refer to the `-O` entry. Symbolic debugging may give confusing results if an optimization level other than zero is selected.

**Note:** Note: Including symbolic debugging information increases the size of the object module.

**Related options**

*-O<level>*, *-gopt*

## 3.2.24. -g77libs

Used on the link line, this option instructs the nvfortran driver to search the necessary g77 or gfortran support libraries to resolve references specific to g77- or gfortran-compiled program units.

**Note:** The g77 or gfortran compiler must be installed on the system on which linking occurs in order for this option to function correctly.

**Default**

The compiler does not search g77 or gfortran support libraries to resolve references at link time.

**Usage**

The following command-line requests that g77 and gfortran support libraries be searched at link time:

```
$ nvfortran -g77libs myprog.f g77_object.o
```

**Description**

Use the `-g77libs` option on the link line if you are linking g77- or gfortran-compiled program units into a nvfortran-compiled main program using the nvfortran driver. When this option is present, the

nvfortran driver searches the necessary g77 and gfortran support libraries to resolve references specific to g77- or gfortran-compiled program units.

**Related options**

*-fortranlibs*

## 3.2.25. –gcc-toolchain=<path>

Specify the gcc toolchain location for use during compilation.

**Default**

Compiles using the default gcc toolchain location (selected during installation).

**Usage**

The following examples compile using the specified gcc 9.3.0 toolchain.

```
$ nvc++ --gcc-toolchain=~/gcc/gcc-9.3.0/  myprog.cpp
```

```
$ nvc++ --gcc-toolchain=~/gcc/gcc-9.3.0/bin/  myprog.cpp
```

```
$ nvc++ --gcc-toolchain=~/gcc/gcc-9.3.0/bin/gcc  myprog.cpp
```

**Description**

The argument can either be gcc root directory, <root directory>/bin, or the gcc executable itself.

**Related options**

None.

## 3.2.26. -gopt

Instructs the compiler to include symbolic debugging information in the object file, and to generate optimized code identical to that generated when -g is not specified.

**Default**

The compiler does not put debugging information into the object module.

**Usage**

In the following example, the object file `myprog.o` contains symbolic debugging information.

```
$ nvfortran -c -gopt myprog.f
```

**Description**

Using `-g` alters how optimized code is generated in ways that are intended to enable or improve debugging of optimized code. The `-gopt` option instructs the compiler to include symbolic debugging information in the object file, and to generate optimized code identical to that generated when -g is not specified.

**Related options**

*-g*, *-M<nvflag>*

## 3.2.27. -gpu

Used in combination with the `-acc`, `-cuda`, `-mp`, and `-stdpar` flags to specify options for GPU code generation. The following sub-options may be used following an equals sign ("="), with multiple sub-options separated by commas:

**`autocompare`**
Automatically compare CPU vs GPU results at execution time: implies redundant

**`ccXY`**
Generate code for a device with compute capability X.Y. Multiple compute capabilities can be specified, and one version will be generated for each. By default, the compiler will detect the compute capability for each installed GPU. Use -help -gpu to see the valid compute capabilities for your installation.

**`ccall`**
Generate code for all compute capabilities supported by this platform and by the selected or default CUDA Toolkit.

**`ccall-major`**
Compile for all major supported compute capabilities.

**`ccnative`**
Detects the visible GPUs on the system and generates codes for them. If no device is available, the compute capability matching NVCC's default will be used.

**`cudaX.Y`**
Use CUDA X.Y Toolkit compatibility, where installed

**`[no]debug`**
Enable [disable] debug information generation in device code

**`deepcopy`**
Enable full deep copy of aggregate data structures in OpenACC; Fortran only

**`fastmath`**
Use routines from the fast math library

**`[no]flushz`**
Enable [disable] flush-to-zero mode for floating point computations on the GPU

**`[no]fma`**
Generate [do not generate] fused multiply-add instructions; default at `-O1`. This is an alias of -M[no]fma.

**`[no]implicitsections`**
Change [do not change] array element references in a data clause into an array section. In C++, the `implicitsections` option will change `update device(a[n])` to `update device(a[0:n])`. In Fortran, it will change `enter data copyin(a(n))` to `enter data copyin(a(:n))`. The default behavior, `noimplicitsections`, can also be changed using rc-files; for example, one could add `set IMPLICITSECTIONS=0;` to siterc or another rcfile.

**`[no]interceptdeallocations`**
Intercept [do not intercept] calls to standard library memory deallocations (e.g. `free`) and call the corresponding CUDA memory deallocation version if address is in pinned or managed memory, regular version otherwise.

**`keep`**
Keep the kernel files (.cubin, .ptx, source)

**[no]lineinfo**
Enable [disable] GPU line information generation

**loadcache:{L1|L2}**
Choose what hardware level cache to use for global memory loads; options include the default, L1, or L2

**[no]managed**
Allocate [do not allocate] any dynamically allocated data in CUDA Managed memory. Use `-gpu=nomanaged` with `-stdpar` to prevent that flag's implicit use of `-gpu=managed` when CUDA Managed memory capability is detected. This option is deprecated.

**maxregcount:n**
Specify the maximum number of registers to use on the GPU; leaving this blank indicates no limit

**mem:{separate|managed|unified}**
Select GPU memory mode for the generated binary. This controls CUDA memory capability to be utilised such as separate GPU memory only (`separate`), GPU Managed Memory for the dynamically allocated data (`managed`), or system memory aka full CUDA Unified Memory (`unified`). Use of Managed or Unified Memory facilitates simpler programming by eliminating the need to detect all data to be copied into and outside of the code region executing on the GPU.

**pinned**
Use CUDA Pinned Memory. This option is deprecated.

**ptxinfo**
Print PTX info

**[no]rdc**
Generate [do not generate] relocatable device code.

**redundant**
Redundant CPU/GPU execution

**safecache**
Allow variable-sized array sections in cache directives; compiler assumes they fit into CUDA shared memory

**sm_XY**
Generate code for a device with compute capability X.Y. Multiple compute capabilities can be specified, and one version will be generated for each. By default, the compiler will detect the compute capability for each installed GPU. Use -help -gpu to see the valid compute capabilities for your installation.

**stacklimit:<l>nostacklimit**
Sets the limit (l) of stack variables in a procedure or kernel, in KB. This option is deprecated.

**[no]unified**
Compile [do not compile] for CUDA Unified memory capability, where system memory is accessible from the GPU. This mode utilizes system and managed memory for dynamically allocated data unless explicit behavior is set through `-gpu=[no]managed`. Use `-gpu=nounified` with `-stdpar` to prevent that flag's implicit use of `-gpu=unified` when CUDA Unified memory capability is detected. This option must appear in both the compile and link lines. This option is deprecated.

**[no]unroll**
Enable [disable] automatic inner loop unrolling; default at -03

**zeroinit**
Initialize allocated device memory with zero

**Usage**

In the following example, the compiler generates code for NVIDIA GPUs with compute capabilities 6.0 and 7.0.

```
$ nvfortran -acc -gpu=cc60,cc70 myprog.f
```

The compiler automatically invokes the necessary software tools to create the kernel code and embeds the kernels in the object file.

To link in the appropriate GPU libraries, you must link an OpenACC program with the `-acc` flag, and similarly for -cuda, -mp, or -stdpar.

**DWARF Debugging Formats**

Use the `-g` option to enable generation of full DWARF information on both the host and device; in the absence of other optimization flags, `-g` sets the optimization level to zero. If a `-O` option raises the optimization level to one or higher, only GPU line information is generated in device code even when `-g` is specified. To enforce full DWARF generation for device code at optimization levels above zero, use the `debug` sub-option to `-gpu`. Conversely, to prevent the generation of dwarf information for device code, use the `nodebug` sub-option to `-gpu`. Both `debug` and `nodebug` can be used independently of `-g`.

## 3.2.28. -help

Used with no other options, `-help` displays options recognized by the driver on the standard output. When used in combination with one or more additional options, usage information for those options is displayed to standard output.

**Default**

The compiler does not display usage information.

**Usage**

In the following example, usage information for `-Minline` is printed to standard output.

```
$ nvc -help -Minline
-Minline[=lib:<inlib>|<maxsize>|<func>|except:<func>|name:<func>|maxsize:<n>|
totalsize:<n>|smallsize:<n>|reshape]
                    Enable function inlining
    lib:<inlib>     Use extracted functions from inlib
    <maxsize>       Set maximum function size to inline
    <func>          Inline function func
    except:<func>   Do not inline function func
    name:<func>     Inline function func
    maxsize:<n>     Inline only functions smaller than n
    totalsize:<n>   Limit inlining to total size of n
    smallsize:<n>   Always inline functions smaller than n
    reshape         Allow inlining in Fortran even when array shapes do not
                    match
    pragma          Fortran Only: Inline only those procedures that have the
                    ``!NVF$ INLINE`` pragma on the source line
                    immediately before the procedure's SUBROUTINE or
                    FUNCTION statement.
    -Minline        Inline all functions that were extracted
```

In the following example, usage information for `-help` shows how groups of options can be listed or examined according to function.

```
$ nvc -help -help
-help[=groups|asm|debug|language|linker|opt|other|
overall|phase|prepro|suffix|switch|target|variable]
```

**Description**

Use the -help option to obtain information about available options and their syntax. You can use -help in one of three ways:

- Use `-help` with no parameters to obtain a list of all the available options with a brief one-line description of each.
- Add a parameter to -help to restrict the output to information about a specific option. The syntax for this usage is this:

  ```
  -help <command line option>
  ```

- Add a parameter to -help to restrict the output to a specific set of options or to a building process. The syntax for this usage is this:

  ```
  -help=<subgroup>
  ```

The following table lists and describes the subgroups available with -help.

Table 5: Table 11. Subgroups for -help Option

| Use this -help option | To get this information... |
|---|---|
| `-help=asm` | A list of options specific to the assembly phase. |
| `-help=debug` | A list of options related to debug information generation. |
| `-help=groups` | A list of available switch classifications. |
| `-help=language` | A list of language-specific options. |
| `-help=linker` | A list of options specific to link phase. |
| `-help=opt` | A list of options specific to optimization phase. |
| `-help=other` | A list of other options, such as ANSI conformance pointer aliasing for C. |
| `-help=overall` | A list of options generic to any NVIDIA HPC compiler. |
| `-help=phase` | A list of build process phases and to which compiler they apply. |
| `-help=prepro` | A list of options specific to the preprocessing phase. |
| `-help=suffix` | A list of known file suffixes and to which phases they apply. |
| `-help=switch` | A list of all known options; this is equivalent to usage of -help without any parameter. |
| `-help=target` | A list of options specific to target processor. |
| `-help=variable` | A list of all variables and their current value. They can be redefined on the command line using syntax VAR=VALUE. |

For more examples of -help, refer to ‘Help with Command-line Options.’

**Related options**

*-#*, *-show*, *-V[release_number]*

## 3.2.29. -I

Adds a directory to the search path for files that are included using either the INCLUDE statement or the preprocessor directive #include.

**Default**

The compiler searches only certain directories for included files.

- For gcc-lib includes: `/usr/lib64/gcc-lib`
- For system includes: `/usr/include`

**Syntax**

```
-Idirectory
```

Where directory is the name of the directory added to the standard search path for include files.

**Usage**

In the following example, the compiler first searches the directory `mydir` and then searches the default directories for include files.

```
$ nvfortran -Imydir
```

**Description**

Adds a directory to the search path for files that are included using the INCLUDE statement or the preprocessor directive #include. Use the -I option to add a directory to the list of where to search for the included files. The compiler searches the directory specified by the -I option before the default directories.

The Fortran INCLUDE statement directs the compiler to begin reading from another file. The compiler uses two rules to locate the file:

- If the file name specified in the INCLUDE statement includes a path name, the compiler begins reading from the file it specifies.
- If no path name is provided in the INCLUDE statement, the compiler searches (in order):
  1. Any directories specified using the -I option (in the order specified)
  2. The directory containing the source file
  3. The current directory

  For example, the compiler applies rule (1) to the following statements:

```
INCLUDE '/bob/include/file1' (absolute path name)
INCLUDE '../../file1' (relative path name)
```

  and rule (2) to this statement:

```
INCLUDE 'file1'
```

**Related options**

*-Mnostdinc*

## 3.2.30. -i2, -i4, -i8

(Fortran only) Treat INTEGER and LOGICAL variables as either two, four, or eight bytes.

**Default**

The compiler treats INTERGER and LOGICAL variables as four bytes.

**Usage**

In the following example, using the -i8 switch causes the integer variables to be treated as 64 bits.

```
$ nvfortran -i8 int.f
```

`int.f` is a function similar to this:

```
int.f
     print *, "Integer size:", bit_size(i)
     end
```

**Description**

Use this option to treat INTEGER and LOGICAL variables as either two, four, or eight bytes. INTEGER*8 values not only occupy 8 bytes of storage, but operations use 64 bits, instead of 32 bits.

- -i2: Treat INTEGER variables as 2 bytes.
- -i4: Treat INTEGER variables as 4 bytes.
- -i8: Treat INTEGER and LOGICAL variables as 8 bytes and use 64-bits for INTEGER*8 operations.

**Related options**

None.

## 3.2.31. -K<flag>

Requests that the compiler provide special compilation semantics with regard to conformance to IEEE 754.

**Default**

The default is `-Knoieee` and the compiler does not provide special compilation semantics.

**Syntax**

-K<flag>

Where flag is one of the following:

**`ieee`**
: Perform floating-point operations in strict conformance with the IEEE 754 standard. Some optimizations are disabled, and on some systems a more accurate math library is linked if `-Kieee` is used during the link step.

**noieee**
Default flag. Use the fastest available means to perform floating-point operations, link in faster non-IEEE libraries if available, and disable underflow traps.

**PIC or pic**
Generate position-independent code. Equivalent to `-fpic`. Provided for compatibility with other compilers.

**trap=option[,option]...**
Controls the behavior of the processor when floating-point exceptions occur. Possible options include:

- `fp`
- `align` (ignored)
- `inv`
- `denorm`
- `divz`
- `ovf`
- `unf`
- `inexact`
- `none`

---

**Note:** Same floating-point exception functionality as `-Ktrap=option[,option]` can be achieved with the runtime environment variable `NVCOMPILER_FPU_STATE`

---

### Usage

In the following example, the compiler performs floating-point operations in strict conformance with the IEEE 754 standard

```
$ nvfortran -Kieee myprog.f
```

### Description

Use `-K` to instruct the compiler to provide special compilation semantics.

`-Ktrap` is only processed by the compilers when compiling main functions or programs. The options `inv`, `denorm`, `divz`, `ovf`, `unf`, and `inexact` correspond to the processor's exception mask bits: invalid operation, denormalized operand, divide-by-zero, overflow, underflow, and precision, respectively.

Normally, the processor's exception mask bits are *on*, meaning that floating-point exceptions are masked – the processor recovers from the exceptions and continues. If a floating-point exception occurs and its corresponding mask bit is *off*, or "unmasked", execution terminates with an arithmetic exception (C's SIGFPE signal).

`-Ktrap=fp` is equivalent to `-Ktrap=inv,divz,ovf`.

---

**Note:** The NVIDIA HPC compilers do not support exception-free execution for `-Ktrap=inexact`. The purpose of this hardware support is for those who have specific uses for its execution, along with the appropriate signal handlers for handling exceptions it produces. It is not designed for normal floating point operation code support.

---

**Related options**

None.

## 3.2.32. -L

Specifies a directory to search for libraries.

**Note:** Multiple -L options are valid. However, the position of multiple -L options is important relative to -l options supplied.

**Default**

The compiler searches the standard library directory.

**Syntax**

```
-Ldirectory
```

Where `directory` is the name of the library directory.

**Usage**

In the following example, the library directory is `/lib` and the linker links in the standard libraries required by NVFORTRAN from this directory.

```
$ nvfortran -L/lib myprog.f
```

In the following example, the library directory `/lib` is searched for the library file `libx.a` and both the directories `/lib` and `/libz` are searched for `liby.a`.

```
$ nvfortran -L/lib -lx -L/libz -ly myprog.f
```

**Description**

Use the -L option to specify a directory to search for libraries. Using -L allows you to add directories to the search path for library files.

**Related options**

*-l*

## 3.2.33. -l<library>

Instructs the linker to load the specified library. The linker searches <library>in addition to the standard libraries.

**Note:** The linker searches the libraries specified with -l in order of appearance *before* searching the standard libraries.

**Syntax**

```
-llibrary
```

Where library is the name of the library to search.

Usage: In the following example, if the standard library directory is `/lib` the linker loads the library `/lib/libmylib.a`, in addition to the standard libraries.

```
$ nvfortran myprog.f -lmylib
```

**Description**

Use this option to instruct the linker to load the specified library. The compiler prepends the characters lib to the library name and adds the .a extension following the library name. The linker searches each library specified before searching the standard libraries.

**Related options**

*-L*

## 3.2.34. -M

Generate make dependence lists. You can use `-MD,filename` (nvc++ only) to generate make dependence lists and print them to the specified file.

## 3.2.35. -M<nvflag>

Selects options for code generation. The options are divided into the following categories:

| | | |
|---|---|---|
| Code generation | Fortran Language Controls | Optimization |
| Environment | C/C++ Language Controls | Miscellaneous |
| Inlining | | |

The following table lists and briefly describes the options alphabetically and includes a field showing the category. For more details about the options as they relate to these categories, refer to '-M Options by Category'.

| nvflag | Description |
|---|---|
| `allocatable=95\|03` | Controls whether to use Fortran 95 or Fortran 2003 semantics in allocatable array as |
| anno | Annotate the assembly code with source code. |
| `[no]autoinline` | When a C/C++ function is declared with the inline keyword, inline it at -O2. |
| `[no]asmkeyword` | Specifies whether the compiler allows the asm keyword in C/C++ source files (nvc an |
| `[no]backslash` | Determines how the backslash character is treated in quoted strings (nvfortran only) |
| `[no]bounds` | Specifies whether array bounds checking is enabled or disabled. |

| nvflag | Description |
|---|---|
| [no]builtin | Do/don't compile with math subroutine builtin support, which causes selected math l |
| byteswapio | Swap byte-order (big-endian to little-endian or vice versa) during I/O of Fortran unfor |
| cache_align | Where possible, align data objects of size greater than or equal to 16 bytes on cache- |
| chkptr | Check for NULL pointers (nvfortran only). |
| chkstk | Check the stack for available space upon entry to and before the start of a parallel re |
| concur | Enable auto-concurrentization of loops. Multiple processors or cores will be used to e |
| cpp | Run the NVIDIA cpp-like preprocessor without performing subsequent compilation st |
| cray | Force Cray Fortran (CF77) compatibility (nvfortran only). |
| cuda | Enables CUDA Fortran. |
| [no]daz | Do/don't treat denormalized operands as zero (default). |
| [no]dclchk | Determines whether all program variables must be declared (nvfortran only). |
| [no]defaultunit | Determines how the asterisk character ("*") is treated in relation to standard input an |
| [no]depchk | Checks for potential data dependencies. |
| [no]dse | Enables [disables] dead store elimination phase for programs making extensive use o |
| [no]dlines | Determines whether the compiler treats lines containing the letter "D" in column one |
| dollar,char | Specifies the character (char) to which the compiler maps the dollar sign symbol (nvf |
| [no]dwarf | Specifies [not] to add DWARF debug information. |
| dwarf2 | When used with -g, generate DWARF2 format debug information. |
| dwarf3 | When used with -g, generate DWARF3 format debug information. |
| extend | Instructs the compiler to accept 132-column source code; otherwise it accepts 72-co |
| extract | invokes the function extractor. |
| [no]fprelaxed[=option] | Perform certain floating point intrinsic functions using relaxed precision. |
| fixed | Instructs the compiler to assume F77-style fixed format source code (nvfortran only) |
| [no]flushz | Do [not] treat denormalized results as zero (default). |
| [no]fpapprox | Specifies not to use low-precision fp approximation operations. |
| free | Instructs the compiler to assume F90-style free format source code (nvfortran only). |
| func32 | The compiler aligns all functions to 32-byte boundaries. |
| gccbug[s] | Matches behavior of certain gcc bugs |
| info | Prints informational messages regarding optimization and code generation to standa |
| inform | Specifies the minimum level of error severity that the compiler displays. |
| inline | Invokes the function inliner. |
| instrument | Generates additional code to enable instrumentation of functions. |

| nvflag | Description |
|---|---|
| [no]iomutex | Determines whether critical sections are generated around Fortran I/O calls (nvfortra |
| [no]ipa | Invokes interprocedural analysis and optimization. |
| keepasm | Preserve intermediate assembly language file. |
| [no]large_arrays | Enables support for 64-bit indexing and single static data objects of size larger than |
| list | Specifies whether the compiler creates a listing file. |
| [no]loop32 | Aligns [does not align] innermost loops on 32-byte boundaries. |
| [no]lre | Enable [disable] loop-carried redundancy elimination. |
| [no]m128 | Recognizes [ignores] __m128, __m128d, and __m128i datatypes. (nvc only) |
| fcon | Instructs the compiler to treat floating-point constants as float data types rather tha |
| neginfo | Instructs the compiler to produce information on why certain optimizations are not p |
| noframe | Eliminates operations that set up a true stack frame pointer for functions. |
| [no]i4 | [do not] treat INTEGER variables and constants as INTEGER(KIND=4). |
| nomain | When the link step is called, don't include the object file that calls the Fortran main p |
| norpath | On Linux, do not add -rpath paths to the link line. |
| [no]stddef | Instructs the compiler to not recognize the standard preprocessor macros. |
| nostdinc | Instructs the compiler to not search the standard location for include files. |
| nostdlib | Instructs the linker to not link in the standard libraries. |
| [no]onetrip | Determines whether each DO loop executes at least once (nvfortran only). |
| novintr | Disable idiom recognition and generation of calls to optimized vector functions. |
| preprocess | Perform cpp-like preprocessing on assembly language and Fortran input source files. |
| [no]r8 | [do not] treat REAL variables and constants as REAL(KIND=8) (nvfortran only). |
| [no]r8intrinsics | Determines how the compiler treats the intrinsics CMPLX and REAL (nvfortran only). |
| [no]recursive | Allocate [do not allocate] local variables on the stack; this allows recursion. SAVEd, da |
| [no]reentrant | Specifies whether the compiler avoids optimizations that can prevent code from bein |
| [no]ref_externals | [do not] force references to names appearing in EXTERNAL statements (nvfortran or |
| safeptr | Instructs the compiler to override data dependencies between pointers and arrays (n |
| safe_lastval | In the case where a scalar is used after a loop, but is not defined on every iteration of |
| [no]save | Determines whether the compiler assumes that all local variables are subject to the S |
| schar | Specifies signed char for characters (nvc and nvc++ only – also see uchar). |
| [no]second_underscore | Do [do not] add the second underscore to the name of a Fortran global if its name al |
| [no]signextend | Do [do not] extend the sign bit, if it is set. |
| [no]single | Do [do not] convert float parameters to double parameter characters (nvc and nvc++ |

| nvflag | Description |
|---|---|
| `standard` | Causes the compiler to flag source code that does not conform to the ANSI standard |
| `[no]stride0` | Do [do not] generate alternate code for a loop that contains an induction variable wh |
| `uchar` | Specifies unsigned char for characters (nvc and nvc++ only – also see schar). |
| `[no]unixlogical` | [do not] treat any non-zero logical variable as .TRUE. . (nvfortran only). |
| `[no]unroll` | Controls loop unrolling. |
| `[no]upcase` | Determines whether the compiler preserves uppercase letters in identifiers. Fortran |
| `varargs` | Forces Fortran program units to assume calls are to C functions with a varargs type i |
| `[no]vect` | Do [do not] invoke the code vectorizer. |

## 3.2.36. -m

Displays a link map on the standard output.

**Default**

The compiler does not display the link map.

**Usage**

When the following example is executed, nvfortran writes the link map to `stdout`.

```
$ nvfortran -m myprog.f
```

**Description**

Use this option to display a link map.

- On Linux, the map is written to `stdout`.

**Related options**

*-c*, *-o*, *-s*, *-u*

## 3.2.37. -march=<target>

An alias for *-mcpu=<target>[<+extension...>]*. Please see *-mcpu=<target>[<+extension...>]* for details.

**Related options**

*-tp <target>*, *-mcpu=<target>[<+extension...>]*, *-mtune=<target>*, and all `-M<nvflag>` options that control environments, as listed in *Environment Controls*

## 3.2.38. -mcmodel=<size>

Generates code for the requested memory model in the Linux execution environment.

Default: The compiler generates code for the small memory model on Arm and x86-64 targets.

**Usage**

The following command line requests the medium memory model:

```
$ nvfortran -mcmodel=medium myprog.f
```

**Arm Description**

The `tiny` memory model limits the combined area for a user's object or executable to 1MB. The maximum code size is 1MB.

The `small` memory model limits the combined area for a user's object or executable to 4GB. The maximum code size is 2GB.

The `medium` memory model is not supported on Arm. This will automatically select the `large` memory model.

The `large` memory model allows unrestricted data size. The maximum code size is 2GB. `-mcmodel=large` is not compatible with `-fPIC` on Arm systems.

**x86-64 Description**

The `tiny` memory model is not supported on x86-64.

The `small` memory model limits the combined area for a user's object or executable to 2GB. Implies `-Mlarge_arrays` on x86-64 targets.

The `medium` memory model allows unrestricted data size. The maxmium code size is 2GB.

The Linux environment provides `static libxxx.a` archive libraries, that are built both with and without `-fpic`, and `dynamic libxxx.so` shared object libraries that are compiled with `-fpic`. Using the link switch `-mcmodel=medium` implies the `-fpic` switch and utilizes the shared libraries by default.

The large memory model is not supported on x86-64.

**Details**

The `tiny` and `small` code models are the fastest and should be suitable for the majority of programs. The `medium` and `large` code models allow for larger code and data sizes, at the cost of extra instructions. Please see the respective SysV ABI documents for more detail.

**Related options**

*-Mlarge_arrays*

## 3.2.39. -mcpu=<target>[<+extension...>]

Sets the target processor. An optional list of architecture extensions may follow the target processor. Architecture extensions are disabled by prepending `no` to the extension name. For example, `+nocrypto`. Extensions are processed in order, from left-to-right.

**Default**

The NVIDIA HPC compilers produce code specifically targeted to the type of processor on which compilation is performed. In particular, the default is to use all supported instructions wherever possible when compiling on a given system.

The default target processor is auto-selected depending on the processor on which the compilation is performed. You can specify a target processor different than the auto-selected default, but that target must be within the same CPU family as the processor on which compilation is performed. The NVIDIA HPC Compilers support 2 different families of CPUs: x86_64 and 64-bit Arm Server CPUs.

Executables created on a given system without the `-mcpu=` flag may not be usable on previous generation systems. For example, executables created on an Intel Skylake processor may use AVX-512 or other instructions that are not available on earlier Intel processors or certain AMD processors.

**Usage**

In the following example, nvfortran sets the target processor to Arm Neoverse-v2 with Crypto support:

```
$ nvfortran -mcpu=neoverse-v2+crypto myprog.f
```

**Description**

Use this option to set the target architecture. By default, the NVIDIA HPC compilers use all supported instructions wherever possible when compiling on a given system.

Processor-specific optimizations can be specified or limited explicitly by using the `-mcpu` option. Thus, it is possible to create executables that are usable on previous-generation systems.

The following list contains the possible suboptions for `-mcpu` and the processors that each suboption is intended to target.

**x86-64**

**px** generate code that is usable on any x86-64 processor-based system.

**host**
generate code targeted for host processor. Link native version of HPC SDK cpu math library.

**native**
generate code targeted for host processor. Alias for -tp host.

**x86-64-v2**
generate code for the x86-64 microarchitectural level including SSE.

**x86-64-v3**
generate code for the x86-64 microarchitectural level including AVX2.

**x86-64-v4**
generate code for the x86-64 microarchitectural level including some AVX512 extensions.

**bulldozer**
generate code for AMD Bulldozer and compatible processors.

**piledriver**
generate code that is usable on any AMD Piledriver processor-based system.

**bdver3**
generate code for AMD Steamroller and compatible processors.

**bdver4**
generate code for AMD Excavator and compatible processors.

**zen** generate code that is usable on any AMD Zen processor-based system (e.g. Naples, Ryzen).

**zen2**
generate code that is usable on any AMD Zen 2 processor-based system (e.g. Rome, 3rd Gen Ryzen).

**zen3**
generate code that is usable on any AMD Zen 3 processor-based system (e.g. Milan, Ryzen 5000).

**zen4**
generate code that is usable on any AMD Zen 4 processor-based system (e.g. Genoa).

**sandybridge**
generate code for Intel Sandy Bridge and compatible processors.

**haswell**
generate code that is usable on any Intel Haswell processor-based system.

**skylake**
generate code that is usable on an Intel Skylake Xeon processor-based system.

**icelake**
generate code that is usable on an Intel Ice Lake Xeon processor-based system.

**cannonlake**
generate code that is usable on an Intel Cannon Lake Xeon processor-based system.

**cascadelake**
generate code that is usable on an Intel Cascade Lake Xeon processor-based system.

**cooperlake**
generate code that is usable on an Intel Cooper Lake Xeon processor-based system.

**tigerlake**
generate code that is usable on an Intel Tiger Lake Xeon processor-based system.

**alderlake**
generate code that is usable on an Intel Alder Lake Xeon processor-based system.

**rocketlake**
generate code that is usable on an Intel Rocket Lake Xeon processor-based system.

**sapphirerapids**
generate code that is usable on an Intel Sapphire Rapids Xeon processor-based system.

**graniterapids**
generate code that is usable on an Intel Granite Rapids Xeon processor-based system.

**Arm**

**px** generate code that is usable on any Arm processor-based system.

**host**
generate code targeted for host processor. Link native version of HPC SDK cpu math library.

**native**
generate code targeted for host processor. Alias for -tp host.

**a64fx**
generate code that is usable on a Fujitsu A64fx processor-based system (SVE x 512).

**neoverse-n1**
generate code that is usable on any Arm Neoverse-N1 processor-based system.

**neoverse-v1**
generate code that is usable on any Arm Neoverse-V1 processor-based system (SVE x 256).

**neoverse-v2**
generate code that is usable on any Arm Neoverse-V2 processor-based system (SVE x 128).

**grace**
generate code that is usable on a NVIDIA Grace processor-based system (SVE x 128).

**graviton3**
generate code that is usable on an AWS Graviton3 processor-based system (SVE x 256).

**graviton4**
generate code that is usable on an AWS Graviton4 processor-based system (SVE x 128).

**thunderx2t99**
generate code that is usable on a Cavium Vulcan processor-based system.

**Related options**

*-tp <target>*, *-march=<target>*, *-mtune=<target>*, and all `-M<nvflag>` options that control environments, as listed in *Environment Controls*

## 3.2.40. -module <moduledir>

Allows you to specify a particular directory in which generated intermediate `.mod` files should be placed.

**Default**

The compiler places `.mod` files in the current working directory, and searches only in the current working directory for pre-compiled intermediate `.mod` files.

**Usage**

The following command line requests that any intermediate module file produced during compilation of `myprog.f` be placed in the directory `mymods`; specifically, the file `./mymods/myprog.mod` is used.

```
$ nvfortran -module mymods myprog.f
```

**Description**

Use the `-module` option to specify a particular directory in which generated intermediate .mod files should be placed. If the `-module <moduledir>` option is present, and USE statements are present in a compiled program unit, then *<moduledir>* is searched for `.mod` intermediate files *prior* to a search in the default local directory.

**Related options**

None.

## 3.2.41. -[no]mp

Enable [disable] OpenMP directives. When enabled, it instructs the compiler to interpret user-inserted OpenMP parallel programming directives and pragmas, and to generate an executable file which will utilize multiple processors in a parallel system.

**Default**

The compiler does not interpret user-inserted OpenMP parallel programming directives and pragmas.

### Usage

The following command line requests processing of any OpenMP directives present in `myprog.f`:

```
$ nvfortran -mp myprog.f
```

### Description

Use the `-mp` option to instruct the compiler to interpret user-inserted OpenMP parallel programming directives and to generate an executable file which utilizes multiple processors in a parallel system.

The suboptions are one or more of the following:

**`[no]align`**
Forces loop iterations to be allocated to OpenMP processes using an algorithm that maximizes alignment of vector sub-sections in loops that are both parallelized and SIMD vectorized. This allocation can improve performance in program units that include many such loops. It can also result in load-balancing problems that significantly decrease performance in program units with relatively short loops that contain a large amount of work in each iteration.

**`[no]autopar`**
Auto-parallelization of loops within `omp loop` is enabled by default. To disable this optimization, use the `noautopar` suboption.

**`gpu`** OpenMP directives are compiled for GPU execution as well as host fallback to the CPU. For target-specific options, refer to the documentation for `-gpu`.

**`multicore`**
OpenMP directives are compiled for multicore CPU execution only; this sub-option is the default.

**`ompt`**
Link against the OMPT-enabled OpenMP runtime library. OMPT is an interface that helps a first-party tool monitor the execution of an OpenMP program.

For more information about how the HPC Compilers support OpenMP, refer to the “Using OpenMP” section of the HPC Compilers User Guide.

### Related options

*-Mconcur[=option [,option,...]]*, *-M[no]vect[=option [,option,...]]*

## 3.2.42. -mtune=<target>

`-mtune=` is provided for compatibility. It performs no operation.

### Related options

*-tp*, *-march=<target>*, *-mcpu=<target>[<+extension...>]*, and all `-M<nvflag` options that control environments, as listed in *Environment Controls*

## 3.2.43. -noswitcherror

Issues warnings instead of errors for unknown switches. Ignores unknown command line switches after printing a warning message.

**Default**

The compiler prints an error message and then halts.

**Usage**

In the following example, the compiler ignores unknown command line switches after printing a warning message.

```
$ nvfortran -noswitcherror myprog.f
```

**Description**

Use this option to instruct the compiler to ignore unknown command line switches after printing a warning message.

---

**Tip:** You can configure this behavior in the `siterc` file by adding: `set NOSWITCHERROR=1`.

---

**Related options**

None.

## 3.2.44. -[no]nvmalloc

Enable [disable] linking with a library containing a custom memory allocator.

**Default**

On Arm, the compiler links in the custom memory allocator library for dynamic allocations.

On x86-64, the compiler links in the system library for dynamic allocations.

**Usage**

In the following example, the compiler uses the custom host memory allocator in place of the system library.

```
$ nvc main.c -nvmalloc
```

**Description**

Use this option to make use of the alternate memory allocator library. Users may see a performance improvement in certain situations by using this library.

**Related options**

None.

## 3.2.45. -O<level>

Invokes code optimization at the specified level.

**Default**

The compiler enables classical global optimization.

**Syntax**

```
-O [level]
```

Where level is an integer from 0 to 4 or “fast”.

**Usage**

In the following example, since no -O option is specified, the compiler sets the optimization to level 1.

```
$ nvfortran myprog.f
```

In the following example, since no optimization level is specified and a -O option is specified, the compiler enables classical global optimizations.

```
$ nvfortran -O myprog.f
```

**Description**

Use this option to invoke code optimization. Using the NVIDIA compiler commands with the -Olevel option (the capital O is for Optimize), you can specify any of the following optimization levels:

**-O0** Level zero specifies no optimization. A basic block is generated for each language statement.

**-O1** Level one specifies local optimization. Scheduling of basic blocks is performed. Register allocation is performed.

**-O** When no level is specified, level global optimizations are performed, including traditional scalar optimizations, induction recognition, and loop invariant motion. No SIMD vectorization is enabled.

**-O2** Level two specifies all level-1 and global optimizations, and enables more advanced optimizations such as SIMD code generation, cache alignment, and partial redundancy elimination.

**-O3** Level three specifies aggressive global optimization. This level performs all level-one and level-two global optimizations and enables more aggressive hoisting and scalar replacement optimizations that may or may not be profitable.

**-Ofast**
Enables -O3, `-Mfprelaxed`, `-Mstack_arrays`, `-Mno-nan`, `-Mno-inf`, and, `-fcx-limited-range`.

---

**Note:** `-Mstack_array` is disabled when both command line options `-Ofast` and `-stdpar` are enabled.

---

**-O4** Level four performs all level-one, level-two, and level-three optimizations and enables hoisting of guarded invariant floating point expressions.

The following table shows the interaction between the -O option, `-g` option, `-Mvect`, and `-Mconcur` options.

Table 7: Table 13. Optimization and -O, -g, -Mvect, and -Mconcur Options

| Optimize Option | Debug Option | -M Option | Optimization Level |
|---|---|---|---|
| none | none | none | 1 |
| none | none | -Mvect | 2 |
| none | none | -Mconcur | 2 |
| none | -g | none | 0 |
| -O | none or -g | none | 2 |
| -Olevel | none or -g | none | level |
| -Olevel < 2 | none or -g | -Mvect | 2 |
| -Olevel < 2 | none or -g | -Mconcur | 2 |

Unoptimized code compiled using the option `-O0` can be significantly slower than code generated at other optimization levels. Like the `-Mvect` option, the `-Munroll` option sets the optimization level to level-2 if no `-O` or `-g` options are supplied. The `-gopt` option is recommended for generation of debug information with optimized code. For more information on optimization, refer to the 'Multicore CPU Optimization' section of the HPC Compilers User Guide.

**Related options**

*-g*, *-M<nvflag>*, *-gopt*

## 3.2.46. -o

Names the executable file. Use the -o option to specify the filename of the compiler object file. The final output is the result of linking.

**Default**

The compiler creates executable filenames as needed. If you do not specify the `-o` option, the default filename is the linker output file `a.out`.

**Syntax**

-o filename

Where *filename* is the name of the file for the compilation output. The *filename* should not have a `.f` extension.

**Usage**

In the following example, the executable file is `myprog` instead of the default `a.out`.

```
$ nvfortran myprog.f -o myprog
```

**Related options**

*-c*, *-E*, *-F*, *-S*

## 3.2.47. -pg

Instructs the compiler to instrument the generated executable for gprof-style `gmon.out` sample-based profiling trace file.

**Default**

The compiler does not instrument the generated executable for gprof-style profiling.

**Usage:**

In the following example the program is compiled for profiling using gprof.

```
$ nvfortran -pg myprog.c
```

**Description**

Use this option to instruct the compiler to instrument the generated executable for gprof-style sample-based profiling. You must use this option at both the compile and link steps. A `gmon.out` style trace is generated when the resulting program is executed, and can be analyzed using gprof.

**Related options**

None.

## 3.2.48. -R<directory>

Instructs the linker to hard-code the pathname <directory> into the search path for generated shared object (dynamically linked library) files.

**Note:** There cannot be a space between R and <directory>.

**Usage**

In the following example, at runtime the a.out executable searches the specified directory, in this case `/home/Joe/myso`, for shared objects.

```
$ nvfortran -R/home/Joe/myso myprog.f
```

**Description**

Use this option to instruct the compiler to pass information to the linker to hard-code the pathname <directory> into the search path for shared object (dynamically linked library) files.

**Related options**

*-fpic*, *-shared*

## 3.2.49. -r

Creates a relocatable object file.

**Default**

The compiler does not create a relocatable object file and does not use the -r option.

**Usage**

In this example, nvfortran creates a relocatable object file.

```
$ nvfortran -r myprog.f
```

**Description**

Use this option to create a relocatable object file.

**Related options**

*-C*, *-O<level>*, *-S*, *-U*

## 3.2.50. -r4 and -r8

(Fortran only) Interprets DOUBLE PRECISION variables as REAL (-r4), or interprets REAL variables as DOUBLE PRECISION (-r8). Note that these options do not override de facto standard type declarations that explicitly declare the number of bytes in the type name (REAL*4 and REAL*8).

**Usage**

In this example, the double precision variables are interpreted as REAL.

```
$ nvfortran -r4 myprog.f
```

**Description**

Interpret DOUBLE PRECISION variables as REAL (-r4) or REAL variables as DOUBLE PRECISION (-r8).

**Related options**

*-i2, -i4, -i8*, *-Mnor8*

## 3.2.51. -rc

Specifies the name of the driver startup configuration file. If the file or pathname supplied is not a full pathname, the path for the configuration file loaded is relative to the $DRIVER path (the path of the currently executing driver). If a full pathname is supplied, that file is used for the driver configuration file.

**Syntax**

```
-rc [path] filename
```

Where path is either a relative pathname, relative to the value of $DRIVER, or a full pathname beginning with “/”. Filename is the driver configuration file.

**Usage**

In the following example, the file `.nvfortranrctest`, relative to `/opt/hpc_sdk/<target>/<release>/compilers/bin`, the value of $DRIVER, is the driver configuration file.

```
$ nvfortran -rc .nvfortranrctest myprog.f
```

**Description**

Use this option to specify the name of the compiler driver startup configuration file. If the file or pathname supplied is not a full pathname, the path for the configuration file loaded is relative to the $DRIVER path – the path of the currently executing compiler driver. If a full pathname is supplied, that file is used for the compiler driver configuration file.

**Related options**

*-show*

## 3.2.52. -S

Stops compilation after the compiling phase and writes the assembly-language output to a file.

**Default**

The compiler does not retain a `.s` file.

**Usage**

In this example, nvfortran produces the file `myprog.s` in the current directory.

```
$ nvfortran -S myprog.f
```

**Description**

Use this option to stop compilation after the compiling phase and then write the assembly-language output to a file. If the input file is `filename.f`, then the output file is `filename.s`.

**Related options**

*-c*, *-E*, *-F*, *-Mkeepasm*, *-o*

## 3.2.53. -s

Strips the symbol-table information from the executable file.

**Default**

The compiler includes all symbol-table information and does not use the -s option.

**Usage**

In this example, nvfortran strips symbol-table information from the `a.out`. executable file.

```
$ nvfortran -s myprog.f
```

**Description**

Use this option to strip the symbol-table information from the executable.

**Related options**

*-c*, *-o*, *-u*

## 3.2.54. -shared

Instructs the compiler to pass information to the linker to produce a shared object (dynamically linked library) file.

**Default**

The compiler does not pass information to the linker to produce a shared object file.

**Usage**

In the following example the compiler passes information to the linker to produce the shared object file:`myso.so`.

```
$ nvfortran -shared myprog.f -o myso.so
```

**Description**

Use this option to instruct the compiler to pass information to the linker to produce a shared object (dynamically linked library) file.

**Related options**

*-fpic*, *-R<directory>*

## 3.2.55. -show

Produces driver help information describing the current driver configuration.

**Default**

The compiler does not show driver help information.

**Usage**

In the following example, the driver displays configuration information to the standard output after processing the driver configuration file.

```
$ nvfortran -show myprog.f
```

**Description**

Use this option to produce driver help information describing the current driver configuration.

**Related options**

*-V[release_number]*, *-v*, *-help*, *-rc*

## 3.2.56. -silent

Do not print warning messages.

**Default**

The compiler prints warning messages.

**Usage**

In the following example, the driver does not display warning messages.

```
$ nvfortran -silent myprog.f
```

**Description**

Use this option to suppress warning messages.

**Related options**

*-v*, *-V[release_number]*, *-w*

## 3.2.57. -soname

The compiler recognizes the -soname option and passes it through to the linker.

**Default**

The compiler does not recognize the -soname option.

**Usage**

In the following example, the driver passes the soname option and its argument through to the linker.

```
$ nvfortran -soname library.so myprog.f
```

**Description**

Use this option to instruct the compiler to recognize the -soname option and pass it through to the linker.

**Related options**

None.

## 3.2.58. -static

Statically link all libraries, including the NVIDIA HPC Compilers runtime.

**Default**

Dynamic linking is the default behavior for Linux

**Usage**

The following command line explicitly compiles for and links to the static version of the NVIDIA HPC Compilers runtime libraries:

```
% nvfortran -static -c object1.f
```

**Description**

You can use this option to explicitly compile for and link to the static versions of the system libraries and NVIDIA HPC Compilers runtime libraries.

**Related options**

*-Bdynamic*, *-static-nvidia*

## 3.2.59. -static-nvidia

Linux only. Compile and statically link only to the NVIDIA HPC Compilers runtime libraries. Other libraries are dynamically linked. Implies `-Mnorpath`.

**Default**

The compiler uses static libraries.

**Usage**

The following command line explicitly compiles for and links to the static version of the NVIDIA HPC Compilers runtime libraries:

```
% nvfortran -static-nvidia -c object1.f
```

**Description**

You can use this option to explicitly compile for and link to the static version of the NVIDIA HPC Compilers runtime libraries.

---

**Note:** On Linux, `-static-nvidia` results in code that runs on most Linux systems without requiring a Portability package.

---

**Related options**

*-Bdynamic*, *-static*

## 3.2.60. -stdpar

Enable ISO C++17 Parallel Algorithms behavior; please refer to -gpu for target-specific options. The supported sub-options may be used following an equals sign ("="), with multiple sub-options separated by commas.

**Default**

Without sub-options, `-stdpar` requests generation of code for execution of C++ Parallel Algorithms on the GPU.

**Sub-options**

**`gpu`** Execute C++ Parallel Algorithms on the GPU; the default.

**`multicore`**
Execute C++ Parallel Algorithms in parallel on the CPU.

**Usage**

The following command-line enables parallelization of C++ Parallel Algorithms for offloading to a GPU.

```
$ nvc++ -stdpar myprog.cpp
```

## 3.2.61. -target

Select the target device for all parallel programming paradigms used (OpenACC, OpenMP, Standard Languages). The following suboptions may be used following an equals sign ("="), with multiple sub-options separated by commas:

**gpu** Globally set the target device to an NVIDIA GPU.

**multicore**
Globally set the target device to a multicore CPU.

**Usage**

The following command-line enables parallelization of C++17 Parallel Algorithms and OpenACC, and globally designates the target device as an NVIDIA GPU.

```
$ nvc++ -stdpar -acc -target=gpu myprog.cpp
```

## 3.2.62. -time

Print execution times for various compilation steps.

**Default**

The compiler does not print execution times for compilation steps.

**Usage**

In the following example, nvfortran prints the execution times for the various compilation steps.

```
$ nvfortran -time myprog.f
```

**Description**

Use this option to print execution times for various compilation steps.

**Related options**

*-#*

## 3.2.63. -tp <target>

Sets the target processor.

**Default**

The NVIDIA HPC compilers produce code specifically targeted to the type of processor on which compilation is performed. In particular, the default is to use all supported instructions wherever possible when compiling on a given system.

The default target processor is auto-selected depending on the processor on which the compilation is performed. You can specify a target processor different than the auto-selected default, but that target must be within the same CPU family as the processor on which compilation is performed. The NVIDIA HPC Compilers support 2 different families of CPUs: x86_64 and 64-bit Arm Server CPUs.

Executables created on a given system without the `-tp` flag may not be usable on previous generation systems. For example, executables created on an Intel Skylake processor may use AVX-512 or other instructions that are not available on earlier Intel processors or certain AMD processors.

**Usage**

In the following example, nvfortran sets the target processor to an Intel Skylake Xeon processor:

```
$ nvfortran -tp=skylake myprog.f
```

**Description**

Use this option to set the target architecture. By default, the NVIDIA HPC compilers use all supported instructions wherever possible when compiling on a given system.

Processor-specific optimizations can be specified or limited explicitly by using the `-tp` option. Thus, it is possible to create executables that are usable on previous-generation systems.

The following list contains the possible suboptions for `-tp` and the processors that each suboption is intended to target.

**px** generate code that is usable on any x86-64 processor-based system.

**host**
generate code targeted for host processor. Link native version of HPC SDK cpu math library.

**native**
generate code targeted for host processor. Alias for -tp host.

**x86-64-v2**
generate code for the x86-64 microarchitectural level including SSE.

**x86-64-v3**
generate code for the x86-64 microarchitectural level including AVX2.

**x86-64-v4**
generate code for the x86-64 microarchitectural level including some AVX512 extensions.

**bulldozer**
generate code for AMD Bulldozer and compatible processors.

**piledriver**
generate code that is usable on any AMD Piledriver processor-based system.

**bdver3**
generate code for AMD Steamroller and compatible processors.

**bdver4**
generate code for AMD Excavator and compatible processors.

**zen** generate code that is usable on any AMD Zen processor-based system (e.g. Naples, Ryzen).

**zen2**
generate code that is usable on any AMD Zen 2 processor-based system (e.g. Rome, 3rd Gen Ryzen).

**zen3**
generate code that is usable on any AMD Zen 3 processor-based system (e.g. Milan, Ryzen 5000).

**zen4**
generate code that is usable on any AMD Zen 4 processor-based system (e.g. Genoa).

**sandybridge**
generate code for Intel Sandy Bridge and compatible processors.

**haswell**
generate code that is usable on any Intel Haswell processor-based system.

**skylake**
generate code that is usable on an Intel Skylake Xeon processor-based system.

**icelake**
generate code that is usable on an Intel Ice Lake Xeon processor-based system.

**cannonlake**
generate code that is usable on an Intel Cannon Lake Xeon processor-based system.

**cascadelake**
generate code that is usable on an Intel Cascade Lake Xeon processor-based system.

**cooperlake**
generate code that is usable on an Intel Cooper Lake Xeon processor-based system.

**tigerlake**
generate code that is usable on an Intel Tiger Lake Xeon processor-based system.

**alderlake**
generate code that is usable on an Intel Alder Lake Xeon processor-based system.

**rocketlake**
generate code that is usable on an Intel Rocket Lake Xeon processor-based system.

**sapphirerapids**
generate code that is usable on an Intel Sapphire Rapids Xeon processor-based system.

**graniterapids**
generate code that is usable on an Intel Granite Rapids Xeon processor-based system.

**host**
generate code targeted for host processor. Link native version of HPC SDK cpu math library.

**native**
generate code targeted for host processor. Alias for -tp host.

**a64fx**
generate code that is usable on a Fujitsu A64fx processor-based system (SVE x 512).

**neoverse-n1**
generate code that is usable on any Arm Neoverse-N1 processor-based system.

**neoverse-v1**
generate code that is usable on any Arm Neoverse-V1 processor-based system (SVE x 256).

**neoverse-v2**
generate code that is usable on any Arm Neoverse-V2 processor-based system (SVE x 128).

**grace**
generate code that is usable on a NVIDIA Grace processor-based system (SVE x 128).

**graviton3**
generate code that is usable on an AWS Graviton3 processor-based system (SVE x 256).

**graviton4**
generate code that is usable on an AWS Graviton4 processor-based system (SVE x 128).

**thunderx2t99**
generate code that is usable on a Cavium Vulcan processor-based system.

**Related options**

All `-M<nvflag>` options that control environments, as listed in *Environment Controls*

## 3.2.64. -[no]traceback

Adds debug information for runtime traceback for use with the environment variable `NVCOMPILER_TERM`.

**Default**

The compiler enables traceback for FORTRAN and disables traceback for C and C++.

**Syntax**

```
-traceback
```

**Usage**

In this example, nvfortran enables traceback for the program `myprog.f`.

```
$ nvfortran -traceback myprog.f
```

**Description**

Use this option to enable or disable runtime traceback information for use with the environment variable `NVCOMPILER_TERM`.

Setting `set TRACEBACK=OFF; `` in ``siterc` or `.mynv*rc` also disables default traceback.

Using ON instead of OFF enables default traceback.

**Related options**

None.

## 3.2.65. -U

Undefines a preprocessor macro.

**Syntax**

```
-Usymbol
```

Where *symbol* is a symbolic name.

**Usage**

The following examples undefine the macro test.

```
$ nvfortran -Utest myprog.F
$ nvfortran -Dtest -Utest myprog.F
```

**Description**

Use this option to undefine a preprocessor macro. You can also use the #undef pre-processor directive to undefine macros.

**Related options**

*-D*, *-M[no]stddef*

## 3.2.66. -u

Initializes the symbol-table with <symbol>, which is undefined for the linker. An undefined symbol triggers loading of the first member of an archive library.

**Default**

The compiler does not use the -u option.

**Syntax**

```
-usymbol
```

Where *symbol* is a symbolic name.

**Usage**

In this example, nvfortran initializes symbol-table with `test`.

```
$ nvfortran -utest myprog.f
```

**Description**

Use this option to initialize the symbol-table with <symbol>, which is undefined for the linker. An undefined symbol triggers loading of the first member of an archive library.

**Related options**

*-c*, *-o*, *-s*

## 3.2.67. -V[release_number]

Displays additional information, including version messages. Further, if a `release_number` is appended, the compiler driver attempts to compile using the specified release instead of the default release.

**Note:** There can be no space between `-V` and `release_number`.

**Default**

The compiler does not display version information and uses the release specified by your path to compile.

**Usage**

The following command-line shows the output using the `-V` option.

```
% nvfortran -V myprog.f
```

The following command-line causes nvc to compile using the 20.7 release instead of the default release.

```
% nvc -V20.7 myprog.c
```

**Description**

Use this option to display additional information, including version messages or, if a release_number is appended, to instruct the compiler driver to attempt to compile using the specified release instead of the default release.

The specified release must be co-installed with the default release

**Related options**

*-Minfo[=option [,option,...]]*, *-v*

## 3.2.68. -v

Displays the invocations of the compiler, assembler, and linker.

**Default**

The compiler does not display individual phase invocations.

**Usage**

In the following example you use -v to see the commands sent to compiler tools, assembler, and linker.

```
$ nvfortran -v myprog.f90
```

**Description**

Use the -v option to display the invocations of the compiler, assembler, and linker. These invocations are command lines created by the compiler driver from the files and the -W options you specify on the compiler command-line.

**Related options**

*-dryrun*, *-Minfo[=option [,option,...]]*, *-V[release_number]*, *-W*

## 3.2.69. -W

Passes arguments to a specific phase.

**Syntax**

```
-W{0 | a | l },option[,option...]
```

**Note:** You cannot have a space between the -W and the single-letter pass identifier, between the identifier and the comma, or between the comma and the option.

**0** (the number zero) specifies the compiler.

**a** specifies the assembler.

**l** (lowercase letter l) specifies the linker.

**option**
is a string that is passed to and interpreted by the compiler, assembler or linker. Options separated by commas are passed as separate command line arguments.

**Usage**

In the following example the linker loads the text segment at address `0xffc00000` and the data segment at address `0xffe00000`.

```
$ nvfortran -Wl,-k,-t,0xffc00000,-d,0xffe00000 myprog.f
```

**Description**

Use this option to pass arguments to a specific phase. You can use the -W option to specify options for the assembler, compiler, or linker.

A given NVIDIA HPC compiler command invokes the compiler driver, which parses the command-line, and generates the appropriate commands for the compiler, assembler, and linker.

**Related options**

*-Minfo[=option [,option,...]]*, *-V[release_number]*, *-v*

## 3.2.70. -Werror

Turn all warning messages into error messages. Compilation fails when errors are detected.

**Default**

The compiler does not abort compilation when only warning messages are generated.

**Usage**

In the following example all warning messages are fatal messages.

```
$ nvfortran -Werror myprog.f
```

**Description**

Use the `-Werror` option to abort compilation upon encountering any warning message.

**Related options**

*-fmax-errors*

## 3.2.71. -w

Do not print warning messages.

**Default**

The compiler prints warning messages.

**Usage**

In the following example no warning messages are printed.

```
$ nvfortran -w myprog.f
```

**Description**

Use the -w option to inhibit warning messages.

**Related options**

*-silent*

## 3.2.72. -Xs

Use legacy standard mode for C and C++.

**Default**

None.

**Usage**

In the following example the compiler uses legacy standard mode.

```
$ nvc -Xs myprog.c
```

**Description**

Use this option to use legacy standard mode for C and C++. Further, this option implies -alias=traditional.

**Related options**

*-alias*, *-Xt*

## 3.2.73. -Xt

Use legacy transitional mode for C and C++.

**Default**

None.

**Usage**

In the following example the compiler uses legacy transitional mode.

```
$ nvc -Xt myprog.c
```

**Description**

Use this option to use legacy transitional mode for C and C++. Further, this option implies -alias=traditional.

**Related options**

*-alias*, *-Xs*

### 3.2.74. -Xlinker

Pass options to the linker.

**Syntax**

```
-Xlinker option[,option...]
```

**Default**

None.

**Usage**

In the following example the option –trace-symbol=foo is passed to the linker, which will cause the Linux linker to list all the files that reference symbol foo.

```
$ nvc -Xlinker --trace-symbol=foo myprog.c
```

**Description**

Use this option pass options to the linker. This is useful when the link step needs to be customized but the compiler doesn't understand the necessary linker options. The options supported by the linker are platform dependent and are not listed here. This option has the same effect as -Wl.

**Related options**

*-W*

## 3.3. C++ and C-specific Compiler Options

There are a large number of compiler options specific to the NVC++ and NVC compilers, especially NVC++. This section provides details on several of these options, but is not exhaustive. For a complete list of available options, including an exhaustive list of NVC++ options, use the `-help` command-line option. For further detail on a given option, use `-help` and specify the option explicitly as described in *-help*.

### 3.3.1. -A

(nvc++ only) Instructs the NVC++ compiler to accept code conforming to the ISO C++ standard, issuing errors for non-conforming code.

**Default**

By default, the compiler accepts code conforming to the standard C++ Annotated Reference Manual.

**Usage**

The following command-line requests ISO conforming C++.

```
$ nvc++ -A hello.cc
```

**Description**

Use this option to instruct the NVC++ compiler to accept code conforming to the ISO C++ standard and to issues errors for non-conforming code.

**Related options**

*-a*

## 3.3.2. -a

(nvc++ only) Instructs the NVC++ compiler to accept code conforming to the ISO C++ standard, issuing warnings for non-conforming code.

**Default**

By default, the compiler accepts code conforming to the standard C++ Annotated Reference Manual.

**Usage**

The following command-line requests ISO conforming C++, issuing warnings for non-conforming code.

```
$ nvc++ -a hello.cc
```

**Description**

Use this option to instruct the NVC++ compiler to accept code conforming to the ISO C++ standard and to issues warnings for non-conforming code.

**Related options**

*-A*

## 3.3.3. -alias

Select optimizations based on type-based pointer alias rules in C and C++.

**Syntax**

```
-alias=[ansi|traditional]
```

**Default**

None.

**Usage**

The following command-line enables optimizations.

```
$ nvc++ -alias=ansi hello.cc
```

**Description**

Use this option to select optimizations based on type-based pointer alias rules in C and C++.

**ansi** Enable optimizations using ANSI C type-based pointer disambiguation

**traditional**
Disable type-based pointer disambiguation

**Related options**

*-Xt*

## 3.3.4. –[no_]alternative_tokens

(nvc++ only) Enables or disables recognition of alternative tokens. These are tokens that make it possible to write C++ without the use of the comma (,) , [, ], #, &, ^, and characters. The alternative tokens include the operator keywords (e.g., and, bitand, etc.) and digraphs.

**Default**

The default behavior is –no_alternative_tokens, that is, to disable recognition of alternative tokens.

**Usage**

The following command-line enables alternative token recognition.

```
$ nvc++ --alternative_tokens hello.cc
```

**Related options**

None.

## 3.3.5. -B

(nvc and nvc++) Enables use of C++ style comments starting with // in C program units.

**Default**

The NVC C compiler does not allow C++ style comments.

**Usage**

In the following example the compiler accepts C++ style comments.

```
$ nvc -B myprog.cc
```

**Description**

Use this option to enable use of C++ style comments starting with // in C program units.

**Related options**

*-Mcpp[=option [,option,...]]*

## 3.3.6. –[no_]bool

(nvc++ only) Enables or disables recognition of bool.

**Default**

The compile recognizes bool: –bool.

**Usage**

In the following example, the compiler does not recognize bool.

```
$ nvc++ --no_bool myprog.cc
```

**Description**

Use this option to enable or disable recognition of bool.

**Related options**

None.

## 3.3.7. –[no_]builtin

Compile with or without math subroutine builtin support.

**Default**

The default is to compile with math subroutine support: –builtin.

**Usage**

In the following example, the compiler does not build with math subroutine support.

```
$ nvc++ --no_builtin myprog.cc
```

**Description**

Use this option to enable or disable compiling with math subroutine builtin support. When you compile with math subroutine builtin support, the selected math library routines are inlined.

**Related options**

None.

## 3.3.8. –[no_]compress_names

Compresses long function names in the file.

**Default**

The compiler does not compress names: –no_compress_names.

**Usage**

In the following example, the compiler compresses long function names.

```
$ nvc++ --compress_names myprog.cc
```

**Description**

Use this option to specify to compress long function names. Highly nested template parameters can cause very long function names. These long names can cause problems for older assemblers. Users encountering these problems should compile all C++ code, including library code with `--compress_names`. Libraries supplied by NVIDIA work with –compress_names.

**Related options**

None.

## 3.3.9. –diag_error <number>

(nvc++ only) Overrides the normal severity of the specified diagnostic messages.

**Default**

The compiler does not override normal diagnostics severity.

**Description**

Use this option to override the normal severity of the specified diagnostic messages and have them treated as errors. The message(s) may be specified using a mnemonic tag or using a diagnostic number.

**Related options**

*–diag_remark <number>*, *–diag_suppress <number>*, *–diag_warning <number>*, *–display_error_number*

## 3.3.10. –diag_remark <number>

(nvc++ only) Overrides the normal severity of the specified diagnostic messages.

**Default**

The compiler does not override normal diagnostics severity.

**Description**

Use this option to override the normal severity of the specified diagnostic messages and have them treated as remarks. The message(s) may be specified using a mnemonic tag or using a diagnostic number.

**Related options**

*–diag_error <number>*, *–diag_suppress <number>*, *–diag_warning <number>*, *–display_error_number*

## 3.3.11. –diag_suppress <number>

(nvc++ only) Overrides the normal severity of the specified diagnostic messages.

**Default**

The compiler does not override normal diagnostics severity.

**Usage**

In the following example, the compiler suppresses the specified diagnostic messages.

```
$ nvc++ --diag_suppress error_tag prog.cc
```

**Description**

Use this option to override the normal severity of the specified diagnostic messages and have them suppressed. The message(s) may be specified using a mnemonic tag or using a diagnostic number.

**Related options**

*–diag_error <number>*, *–diag_remark <number>*, *–diag_warning <number>*, *–display_error_number*

## 3.3.12. –diag_warning <number>

(nvc++ only) Overrides the normal severity of the specified diagnostic messages.

**Default**

The compiler does not override normal diagnostics severity.

**Usage**

In the following example, the compiler overrides the severity of the specified diagnostic messages and treats them as warnings.

```
$ nvc++ --diag_warning an_error_tag myprog.cc
```

**Description**

Use this option to override the normal severity of the specified diagnostic messages and have them treated as warnings. The message(s) may be specified using a mnemonic tag or using a diagnostic number.

**Related options**

*–diag_error <number>*, *–diag_remark <number>*, *–diag_suppress <number>*, *–display_error_number*

## 3.3.13. –display_error_number

(nvc++ only) Displays the error message number in any diagnostic messages that are generated. The option may be used to determine the error number to be used when overriding the severity of a diagnostic message.

**Default**

The compiler does not display error message numbers for generated diagnostic messages.

**Usage**

In the following example, the compiler displays the error message number for any generated diagnostic messages.

```
$ nvc++ --display_error_number myprog.cc
```

**Description**

Use this option to display the error message number in any diagnostic messages that are generated. You can use this option to determine the error number to be used when overriding the severity of a diagnostic message.

**Related options**

*–diag_error <number>*, *–diag_remark <number>*, *–diag_suppress <number>*, *–diag_warning <number>*

## 3.3.14. -e<number>

(nvc++ only) Set the C++ front-end error limit to the specified <number>.

## 3.3.15. –no_exceptions

(nvc++ only) Disables exception handling support.

**Default**

Exception handling support is enabled.

**Usage**

In the following example, the compiler does not provide exception handling support.

```
$ nvc++ --no_exceptions myprog.cc
```

**Description**

Use this option to disable exception handling support. When exception handling is turned off, any try/catch blocks or throw expressions in the code will result in a compilation error, and any exception specifications will be ignored.

## 3.3.16. -fvisibility=<visibility>

(nvc++ only) Sets the visibility of ELF symbols.

**Default**

Sets the visibility of ELF symbols.

**Usage**

All symbols are marked with global visibility unless overridden with this switch or with the visibility attribute.

```
$ nvc++ -fvisibility=default hello.cp
```

**Description**

The *visibility* argument can take on one of four values: default, internal, hidden, or protected.

## 3.3.17. –gnu_version <num>

(nvc++ only) Sets the GNU C++ compatibility version.

**Default**

The compiler uses the latest version installed on the system on which compilation is performed.

**Usage**

In the following example, the compiler sets the GNU version to 4.3.4.

```
$ nvc++ --gnu_version 4.3.4 myprog.cc
```

**Description**

Use this option to set the GNU C++ compatibility version to use when you compile.

## 3.3.18. –[no]llalign

(nvc++ only) Enables or disables alignment of long long integers on long long boundaries.

**Default**

The compiler aligns long long integers on long long boundaries: –llalign.

**Usage**

In the following example, the compiler does not align long long integers on long long boundaries.

```
$ nvc++ --nollalign myprog.cc
```

**Description**

Use this option to allow enable or disable alignment of long long integers on long long boundaries.

**Related options**

None.

## 3.3.19. -M

Generates a list of make dependencies and prints them to stdout.

**Note:** Note: The compilation stops after the preprocessing phase.

**Default**

The compiler does not generate a list of make dependencies.

**Usage**

In the following example, the compiler generates a list of make dependencies.

```
$ nvc++ -M myprog.cc
```

**Description**

Use this option to generate a list of make dependencies and print them to stdout.

**Related options**

*-MD[<dfile>]*, *-P*

## 3.3.20. -MD[<dfile>]

Generates a list of make dependencies and prints them to a file.

**Default**

The compiler does not generate a list of make dependencies.

**Usage**

In the following example, the compiler generates a list of make dependencies and prints them to the file myprog.d.

```
$ nvc++ -MD myprog.cc
```

**Description**

Use this option to generate a list of make dependencies and print them to a file. The name of the file is determined by the name of the file under compilation, or is as specified using the optional `<dfile>` argument.

**Related options**

*-M*,*-P*

## 3.3.21. –optk_allow_dollar_in_id_chars

(nvc++ only) Accepts dollar signs ($) in identifiers.

**Default**

The compiler does not accept dollar signs ($) in identifiers.

**Usage**

In the following example, the compiler allows dollar signs ($) in identifiers.

```
$ nvc++ --optk_allow_dollar_in_id_chars myprog.cc
```

**Description**

Use this option to instruct the compiler to accept dollar signs ($) in identifiers.

## 3.3.22. -P

Halts the compilation process after preprocessing and writes the preprocessed output to a file.

**Default**

The compiler produces an executable file.

**Usage**

In the following example, the compiler produces the preprocessed file `myprog.i` in the current directory.

```
$ nvc++ -P myprog.cc
```

**Description**

Use this option to halt the compilation process after preprocessing and write the preprocessed output to a file. If the input file is `filename.c` or `filename.cc.`, then the output file is `filename.i`.

**Related options**

*-C*, *-c*, *-e<number>*, *-Mkeepasm*, *-o*, *-S*

## 3.3.23. –pedantic

Prints warnings from included <system header files>.

**Default**

The compiler does not print warnings from the included system header files.

**Usage**

In the following example, the compiler prints the warnings from the included system header files.

```
$ nvc++ --pedantic myprog.cc
```

**Related options**

None.

## 3.3.24. –preinclude=<filename>

(nvc++ only) Specifies the name of a file to be included at the beginning of the compilation.

In the following example, the compiler includes the file `incl_file.c` at the beginning of the compilation. me

```
$ nvc++ --preinclude=incl_file.c myprog.cc
```

**Description**

Use this option to specify the name of a file to be included at the beginning of the compilation. For example, you can use this option to set system-dependent macros and types.

**Related options**

None.

## 3.3.25. –[no_]using_std

(nvc++ only) Enables or disables implicit use of the std namespace when standard header files are included.

**Default**

The compiler uses std namespace when standard header files are included: –using_std.

**Usage**

The following command-line disables implicit use of the std namespace:

```
$ nvc++ --no_using_std hello.cc
```

**Description**

Use this option to enable or disable implicit use of the std namespace when standard header files are included in the compilation.

**Related options**

*-M[no]stddef*

## 3.3.26. -Xfilename

(nvc++ only) Generates cross-reference information and places output in the specified file.

**Syntax:**

-Xfoo

where `foo` is the specified file for the cross reference information.

**Default**

The compiler does not generate cross-reference information.

**Usage**

In the following example, the compiler generates cross-reference information, placing it in the file: `xreffile`.

```
$ nvc++ -Xxreffile myprog.cc
```

**Description**

Use this option to generate cross-reference information and place output in the specified file. This is an EDG option.

**Related options**

None.

# 3.4. -M Options by Category

This section describes each of the options available with -M by the categories:

| | | | |
|---|---|---|---|
| Code Generation | Fortran Language Controls | Optimization | Environment |
| C/C++ Language Controls | Inlining | Miscellaneous | |

The following sections provide detailed descriptions of several, but not all, of the -M<nvflag> options. For a complete alphabetical list of all the options, refer to *Table 12*. These options are grouped according to categories and are listed with exact syntax, defaults, and notes concerning similar or related options.

For the latest information and description of a given option, or to see all available options, use the -help command-line option, described in *-help*.

### 3.4.1. Code Generation Controls

This section describes the `-M<nvflag>` options that control code generation.

**Default:** For arguments that you do not specify, the default code generation controls are these:

| | | |
|---|---|---|
| daz | norecursive | nosecond_underscore |
| flushz | noreentrant | nostride0 |
| noref_externals | signextend | |

**Related options:** `-D`, `-I`, `-L`, `-l`, `-U`.

The following list provides the syntax for each `-M<nvflag>` option that controls code generation. Each option has a description and, if appropriate, any related options.

**`-Mdaz`**
Set IEEE denormalized floating point operands to zero; there is a performance benefit but misleading results can occur, such as when dividing a small normalized number by a denormalized number. To take effect, this option must be set when compiling the main program/function. [default - x86_64 and aarch64]

---

**Note:** Same functionality can be achieved with the runtime environment variable `NVCOMPILER_FPU_STATE`

---

**`-Mnodaz`**
Do not treat denormalized numbers as zero. To take effect, this option must be set for the main program.

---

**Note:** Same functionality can be achieved with the runtime environment variable `NVCOMPILER_FPU_STATE`

---

**`-Mnodwarf`**
Specifies not to add DWARF debug information.

**`-Mdwarf2`**
Generate DWARF2 format debug information. To take effect, this option must be used in combination with `-g`.

**`-Mdwarf3`**
Generate DWARF3 format debug information. To take effect, this option must be used in combination with `-g`.

**`-Mflushz`**
Set floating point control register to flush-to-zero mode; if a floating-point underflow occurs, the result is set to zero. To take effect, this option must be set when compiling the main program/function. [default - x86_64 and aarch64]

---

**Note:** Same functionality can be achieved with the runtime environment variable `NVCOMPILER_FPU_STATE`

---

**-Mnoflushz**
Do not set flush-to-zero mode; generate underflows. To take effect, this option must be set for the main program.

**Note:** Same functionality can be achieved with the runtime environment variable NVCOMPILER_FPU_STATE

**-Mfma**
Enable FMA (fused multiply-add) generation on both the CPU and GPU; default at -01.

**Note:** The global -Mfma option can be used in conjunction with -gpu=[no]fma to explicitly enable/disable FMAs on either the CPU or GPU.

Example:

```
-Mfma                   // Enable CPU and GPU FMAs.
-Mfma -gpu=nofma        // Enable CPU FMAs and disable GPU FMAs.
-Mnofma -gpu=fma        // Disable CPU FMAs and enable GPU FMAs.
-Mnofma                 // Disable CPU and GPU FMAs.
```

**-Mnofma**
Disable FMA (fused multiply-add) generation on both the CPU and GPU.

**Note:** The global -Mnofma option can be used in conjunction with -gpu=[no]fma to explicitly enable/disable FMAs on either the CPU or GPU.

**-Mfunc32**
Align functions on 32-byte boundaries.

**-Minstrument [=functions]**
Generate additional code to enable instrumentation of functions.

**Note:** The option -Minstrument=functions is the same as -Minstrument.

Just after function entry and just before function exit, the following profiling functions are called with the address of the current function and its call site.

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

**Note:** In these calls, the first argument is the address of the start of the current function.

Implies -Mframe.

**-Minstrument-exclude-file-list=<filelist>**
Instruct the compiler not to instrument functions in files whose path contains <filelist>. Used in conjunction with -Minstrument[=functions]

**-Minstrument-exclude-func-list=<functions>**
Instruct the compiler not to instrument functions that contain the substring <functions>. Used in conjunction with -Minstrument[=functions]

**-Mlarge_arrays**
Enable support for 64-bit indexing and single static data objects larger than 2 GB in size. On x86-64 targets, this option is the default in the presence of `-mcmodel=medium`. It can be used separately together with the default small memory model for certain 64-bit applications that manage their own memory space. For more information, refer to the 'Programming Considerations for 64-Bit Environments' section of the HPC Compilers User Guide

**-Mnolarge_arrays**
Disable support for 64-bit indexing and single static data objects larger than 2 GB in size. On x86-64 targets, when this option is placed after `-mcmodel=medium` on the command line, it disables use of 64-bit indexing for applications that have no single data object larger than 2 GB. For more information, refer to the 'Programming Considerations for 64-Bit Environments' section of the HPC Compilers User Guide.

**-Mnomain**
Instructs the compiler not to include the object file that calls the Fortran main program as part of the link step. This option is useful for linking programs in which the main program is written in C/C++ and one or more subroutines are written in Fortran (Fortran only).

**-M[no]pre**
enables [disables] partial redundancy elimination.

**-Mrecursive**
instructs the compiler to allow Fortran subprograms to be called recursively.

**-Mnorecursive**
Fortran subprograms may not be called recursively.

**-Mref_externals**
force references to names appearing in EXTERNAL statements (Fortran only).

**-Mnoref_externals**
do not force references to names appearing in EXTERNAL statements (Fortran only).

**-Mreentrant**
instructs the compiler to avoid optimizations that can prevent code from being reentrant.

**-Mnoreentrant**
instructs the compiler not to avoid optimizations that can prevent code from being reentrant.

**-Msecond_underscore**
instructs the compiler to add a second underscore to the name of a Fortran global symbol if its name already contains an underscore. This option is useful for maintaining compatibility with object code compiled using gfortran, which uses this convention by default (Fortran only).

**-Mnosecond_underscore**
instructs the compiler not to add a second underscore to the name of a Fortran global symbol if its name already contains an underscore (Fortran only).

**-Msafe_lastval**
When a scalar is used after a loop, but is not defined on every iteration of the loop, the compiler does not by default parallelize the loop. However, this option tells the compiler it's safe to parallelize the loop. For a given loop, the last value computed for all scalars makes it safe to parallelize the loop.

**-Msignextend**
instructs the compiler to extend the sign bit that is set as a result of converting an object of one data type to an object of a larger signed data type.

**-Mnosignextend**
instructs the compiler not to extend the sign bit that is set as the result of converting an object

of one data type to an object of a larger data type.

**-Mstack_arrays**
places automatic arrays on the stack.

**-Mnostack_arrays**
allocates automatic arrays on the heap. -Mnostack_arrays is the default and what traditionally has been the approach used.

**-Mstride0**
instructs the compiler to inhibit certain optimizations and to allow for stride 0 array references. This option may degrade performance and should only be used if zero-stride induction variables are possible.

**-Mnostride0**
instructs the compiler to perform certain optimizations and to disallow for stride 0 array references.

**-Mvarargs**
force Fortran program units to assume procedure calls are to C functions with a varargs-type interface (nvfortran only).

## 3.4.2. C/C++ Language Controls

This section describes the -M<nvflag> options that affect C++ and C language interpretations by the NVC++ and NVC compilers. These options are only valid to the nvc++ and nvc compiler drivers.

**Default:** For arguments that you do not specify, the defaults are as follows:

| | |
|---|---|
| noasmkeyword | nosingle |
| dollar,_ | schar |

**Usage:**

In this example, the compiler allows the asm keyword in the source file.

```
$ nvc -Masmkeyword myprog.c
```

In the following example, the compiler maps the dollar sign to the dot character.

```
$ nvc -Mdollar,. myprog.c
```

In the following example, the compiler treats floating-point constants as float values, rather than the default double.

```
$ nvc -Mfcon myprog.c
```

In the following example, the compiler does not convert float parameters to double parameters.

```
$ nvc -Msingle myprog.c
```

Without -Muchar or with -Mschar, the variable ch is a signed character:

```
char ch;
signed char sch;
```

If -Muchar is specified on the command line:

```
$ nvc -Muchar myprog.c
```

char ch in the preceding declaration is equivalent to:

```
unsigned char ch;
```

The following list provides the syntax for each -M<nvflag> option that controls code generation in C++ and C. Each option has a description and, if appropriate, any related options.

**-Masmkeyword**
instructs the compiler to allow the asm keyword in C source files. The syntax of the asm statement is as follows:

```
asm("statement");
```

Where *statement* is a legal assembly-language statement. The quote marks are required.

---

**Note:** The current default is to support gcc's extended asm, where the syntax of extended asm includes asm strings. The -M[no]asmkeyword switch is useful only if the target device is a Pentium 3 or older cpu type (-tp piii|p6|k7|athlon|athlonxp|px).

---

**-Mnoasmkeyword**
instructs the compiler not to allow the asm keyword in C source files. If you use this option and your program includes the asm keyword, unresolved references are generated

**-Mdollar,char**
char specifies the character to which the compiler maps the dollar sign ($). The NVC compiler allows the dollar sign in names; ANSI C does not allow the dollar sign in names.

**-M[no]eh_frame**
instructs the linker to keep eh_frame call frame sections in the executable.

---

**Note:** The eh_frame option is available only on newer Linux systems that supply the system unwind libraries.

---

**-Mfcon**
instructs the compiler to treat floating-point constants as float data types, instead of double data types. This option can improve the performance of single-precision code.

**-M[no]m128**
instructs the compiler to recognize [ignore] __m128, __m128d, and __m128i datatypes.

**-Mschar**
specifies signed char characters. The compiler treats "plain" char declarations as signed char.

**-Msingle**
do not to convert float parameters to double parameters in non-prototyped functions. This option can result in faster code if your program uses only float parameters. However, since ANSI C specifies that routines must convert float parameters to double parameters in non-prototyped functions, this option results in non-ANSI conformant code.

**-Mnosingle**
instructs the compiler to convert float parameters to double parameters in non-prototyped functions.

**-Muchar**
instructs the compiler to treat “plain” char declarations as unsigned char.

## 3.4.3. Environment Controls

This section describes the -M<nvflag> options that control environments.

**Default:** For arguments that you do not specify, the default environment option depends on your configuration.

The following list provides the syntax for each -M<nvflag> option that controls environments. Each option has a description and, if appropriate, a list of any related options.

**-Mnostartup**
instructs the linker not to link in the standard startup routine that contains the entry point (_start) for the program.

**Note:** If you use the -Mnostartup option and do not supply an entry point, the linker issues the following error message: Warning: cannot find entry symbol _start

**-M[no]hugetlb**
links in the huge page runtime library. Enables large 2-megabyte pages to be allocated. The effect is to reduce the number of TLB entries required to execute a program. This option is most effective on newer architectures; older architectures do not have enough TLB entries for this option to be beneficial. By itself, the huge suboption tries to allocate as many huge pages as required. You can also limit the pages allocated by using the environment variable `NVCOMPILER_HUGE_PAGES`.

**-M[no]stddef**
instructs the compiler not to predefine any macros to the preprocessor when compiling a C program.

**-Mnostdinc**
instructs the compiler to not search the standard location for include files.

**-Mnostdlib**
instructs the linker not to link in the standard libraries `libnvf.a`, `libm.a`, `libc.a`, and `libnvc.a` in the library directory `lib` within the standard directory. You can link in your own library with the -l option or specify a library directory with the -L option.

## 3.4.4. Fortran Language Controls

This section describes the `-M<nvflag>` options that affect Fortran language interpretations by the NVIDIA Fortran compiler. These options are valid only for the nvfortran compiler driver.

**Default:** Before looking at all the options, let’s look at the defaults. For arguments that you do not specify, the defaults are as follows:

| | | | | |
|---|---|---|---|---|
| nobackslash | nodefaultunit | dollar,_ | noonetrip | nounixlogical |
| nodclchk | nodlines | noiomutex | nosave | noupcase |

The following list provides the syntax for each -M<nvflag> option that affect Fortran language interpretations. Each option has a description and, if appropriate, a list of any related options.

**-Mallocatable=95|03**
controls whether Fortran 95 or Fortran 2003 semantics are used in allocatable array assignments. The default behavior is to use Fortran 95 semantics; the 03 option instructs the compiler to use Fortran 2003 semantics.

**-Mbackslash**
instructs the compiler to treat the backslash as a normal character, and not as an escape character in quoted strings.

**-Mnobackslash**
instructs the compiler to recognize a backslash as an escape character in quoted strings (in accordance with standard C usage).

**-Mdclchk**
instructs the compiler to require that all program variables be declared.

**-Mnodclchk**
instructs the compiler not to require that all program variables be declared.

**-Mdefaultunit**
instructs the compiler to treat "*" as a synonym for standard input for reading and standard output for writing.

**-Mnodefaultunit**
instructs the compiler to treat "*" as a synonym for unit 5 on input and unit 6 on output.

**-Mdlines**
instructs the compiler to treat lines containing "D" in column 1 as executable statements (ignoring the "D").

**-Mnodlines**
instructs the compiler not to treat lines containing "D" in column 1 as executable statements. The compiler does not ignore the "D".

**-Mdollar,char**
char specifies the character to which the compiler maps the dollar sign. The compiler allows the dollar sign in names.

**-Mextend**
instructs the compiler to accept 132-column source code; otherwise it accepts 72-column code.

**-Mfixed**
instructs the compiler to assume input source files are in FORTRAN 77-style fixed form format.

**-Mfree**
instructs the compiler to assume input source files are in Fortran 90/95 freeform format.

**-Miomutex**
instructs the compiler to generate critical section calls around Fortran I/O statements.

**-Mnoiomutex**
instructs the compiler not to generate critical section calls around Fortran I/O statements.

**-Monetrip**
instructs the compiler to force each DO loop to execute at least once. This option is useful for programs written for earlier versions of Fortran.

**-Mnoonetrip**
instructs the compiler not to force each DO loop to execute at least once.

**-Msave**
instructs the compiler to assume that all local variables are subject to the SAVE statement. This may allow older Fortran programs to run, but it can greatly reduce performance.

**-Mnosave**
instructs the compiler not to assume that all local variables are subject to the SAVE statement.

**-Mstandard**
instructs the compiler to flag non-ANSI-conforming source code.

**-Munixlogical**
directs the compiler to treat logical values as true if the value is non-zero and false if the value is zero (UNIX F77 convention). When `-Munixlogical` is enabled, a logical value or test that is non-zero is `.TRUE.`, and a value or test that is zero is `.FALSE.`. In addition, the value of a logical expression is guaranteed to be one (1) when the result is `.TRUE.`.

**-Mnounixlogical**
directs the compiler to use the VMS convention for logical values for true and false. Even values are true and odd values are false.

**-Mupcase**
instructs the compiler to preserve uppercase letters in identifiers. With `-Mupcase`, the identifiers X and x are different. Keywords must be in lower case. This selection affects the linking process. If you compile and link the same source code using `-Mupcase` on one occasion and `-Mnoupcase` on another, you may get two different executables - depending on whether the source contains uppercase letters. The standard libraries are compiled using the default `-Mnoupcase`.

**-Mnoupcase**
instructs the compiler to convert all identifiers to lower case. This selection affects the linking process. If you compile and link the same source code using `-Mupcase` on one occasion and `-Mnoupcase` on another, you may get two different executables, depending on whether the source contains uppercase letters. The standard libraries are compiled using `-Mnoupcase`.

## 3.4.5. Inlining Controls

This section describes the `-M<nvflag>` options that control function inlining.

**Usage:** Before looking at all the options, let's look at a few examples. In the following example, the compiler extracts functions that have 500 or fewer statements from the source file `myprog.f` and saves them in the file `extract.il`.

```
$ nvfortran -Mextract=500 -o extract.il myprog.f
```

In the following example, the compiler inlines functions with fewer than approximately 100 statements in the source file `myprog.f`.

```
$ nvfortran -Minline=maxsize:100 myprog.f
```

**Related options:** -o, -Mextract

The following list provides the syntax for each -M<nvflag> option that controls function inlining. Each option has a description and, if appropriate, a list of any related options.

**-M[no]autoinline[=option[,option,...]]**
instructs the compiler to inline [not to inline] a C++ and C functions at -O2, where the option can be any of these:

**maxsize:n**
instructs the compiler not to inline functions of size > *n*. The default size is 100.

**nostatic**
do not inline static functions without the inline keyword

**totalsize:n**
instructs the compiler to stop inlining when the size equals *n*. The default size is 800.

**-Mextract[=option[,option,...]]**
Extracts functions from the file indicated on the command line and creates or appends to the specified extract directory where option can be any of the following:

**name:func**
instructs the extractor to extract function *func* from the file.

**pragma**
Fortran Only: instructs the extractor to extract procedures that have the !NVF$ INLINE pragma on a separate source line immediately before the procedure's SUBROUTINE or FUNCTION statement.

**size:number**
instructs the extractor to extract functions with *number* or fewer statements from the file.

**lib:filename.ext**
instructs the extractor to use directory `filename.ext` as the extract directory, which is required to save and re-use inline libraries.

If you specify both name and size, the compiler extracts functions that match func, or that have number or fewer statements. For examples of extracting functions, refer to the 'Using Function Inlining' section of the HPC Compilers User Guide.

**-Minline[=option[,option,...]]**
instructs the compiler to pass options to the function inliner, where the option can be any of the following:

**except:func**
Inlines all eligible functions except `func`, a function in the source text. You can use a comma-separated list to specify multiple functions.

**[name:]func**
Inlines all functions in the source text whose name matches `func`. You can use a comma-separated list to specify multiple functions.

The function name should be a non-numeric string that does not contain a period. You can also use a `name:` prefix followed by the function name. If `name:` is specified, what follows is always the name of a function.

**[maxsize:]number**
A numeric option is assumed to be a size. Functions of size `number` or less are inlined. If both `number` and `function` are specified, then functions matching the given name(s) or meeting the size requirements are inlined.

The size *number* need not exactly equal the number of statements in a selected function; the size parameter is merely a rough gauge.

**[no]reshape**
instructs the inliner to allow [disallow] inlining in Fortran even when array shapes do not match. The default is `-Minline=noreshape`, except with `-Mconcur` or `-mp`, where the default is `-Minline=reshape`.

**smallsize:number**
Always inline functions of size smaller than `number` regardless of other size limits.

**totalsize:number**
Stop inlining in a function when the function's total inlined size reaches the `number` specified.

**[lib:]filename.ext**
instructs the inliner to inline the functions within the library file `filename.ext`. The compiler assumes that a `filename.ext` option containing a period is a library file.

---

**Tip:** Create the library file using the `-Mextract` option. You can also use a `lib:` prefix followed by the library name.

- If `lib:` is specified, no period is necessary in the library name. Functions from the specified library are inlined.
- If no library is specified, functions are extracted from a temporary library created during an extract prepass.

---

If you specify both `func` and `number`, the compiler inlines functions that match the function name or have `number` or fewer statements.

Inlining can be disabled with `-Mnoinline`.

For examples of inlining functions, refer to the 'Using Function Inlining' section of the HPC Compilers User Guide.

## 3.4.6. Optimization Controls

This section describes the -M<nvflag> options that control optimization.

**Default:** Before looking at all the options, let's look at the defaults. For arguments that you do not specify, the default optimization control options are as follows:

| | | | |
|---|---|---|---|
| depchk | noipa | nounroll | nor8 |
| i4 | nolre | novect | nor8intrinsics |
| nofprelaxed | noprefetch | | |

**Usage:** In this example, the compiler invokes the vectorizer with use of packed SIMD instructions enabled.

```
$ nvfortran -Mvect=simd -Mcache_align myprog.f
```

---

**Note:** If you do not supply any sub-options to `-Mvect`, the compiler uses defaults that are dependent upon the target system. Not all sub-options are valid on all target systems.

---

**Related options:** -g, -0

The following list provides the syntax for each `-M<nvflag>` option that controls optimization. Each option has a description and, if appropriate, a list of any related options.

**-Mcache_align**
Align unconstrained objects of length greater than or equal to 16 bytes on cache-line boundaries. An unconstrained object is a data object that is not a member of an aggregate structure or common block. This option does not affect the alignment of allocatable or automatic arrays. To effect cache-line alignment of stack-based local variables, the main program or function must be compiled with `-Mcache_align`.

**-Mconcur[=option [,option,...]]**
Instructs the compiler to enable auto-parallelization of loops for multicore CPUs. If `-Mconcur` is specified, multiple CPU cores will be used to execute loops that the compiler determines to be parallelizable. `option` is one of the following:

**allcores**
Instructs the compiler to use all available cores. Use this option at link time.

**[no]altcode:n**
Instructs the parallelizer to generate alternate serial code for parallelized loops.

- If `altcode` is specified without arguments, the parallelizer determines an appropriate cutoff length and generates serial code to be executed whenever the loop count is less than or equal to that length.
- If `altcode:n` is specified, the serial altcode is executed whenever the loop count is less than or equal to n.
- If `noaltcode` is specified, the parallelized version of the loop is always executed regardless of the loop count.

**cncall**
Indicates that calls in parallel loops are safe to parallelize. Also, no minimum loop count threshold must be satisfied before parallelization will occur, and last values of scalars are assumed to be safe.

**[no]innermost**
Instructs the parallelizer to enable parallelization of innermost loops. The default is to not parallelize innermost loops, since it is usually not profitable on dual-core processors.

**levels:n**
Parallelize loops nested at most n levels deep.

**noassoc**
Instructs the parallelizer to disable parallelization of loops with reductions.

When linking, the `-Mconcur` switch must be specified or unresolved references result.

---

**Note:** This option applies only on shared-memory multi-processor (SMP) or multicore CPU-based systems.

---

**-Mcray[=option[,option,...]]**
(Fortran only) Force Cray Fortran compatibility with respect to the listed options. Possible values of option include:

**pointer**
for purposes of optimization, it is assumed that pointer-based variables do not overlay the storage of any other variable.

**-Mdepchk**
instructs the compiler to assume unresolved data dependencies actually conflict.

**-Mnodepchk**
Instructs the compiler to assume potential data dependencies do not conflict. However, if data dependencies exist, this option can produce incorrect code.

**-Mdse**
Enables a dead store elimination phase that is useful for programs that rely on extensive use of inline function calls for performance. This is disabled by default.

**-Mnodse**
Disables the dead store elimination phase. This is the default.

**-M[no]fpapprox [=option]**
Perform certain floating point operations using low-precision approximation. `-Mnofpapprox` specifies not to use low-precision fp approximation operations. By default `-Mfpapprox` is not used. If `-Mfpapprox` is used without suboptions, it defaults to use approximate `div`, `sqrt`, and `rsqrt`. The available suboptions are these:

**div**
Approximate floating point division

**sqrt**
Approximate floating point square root

**rsqrt**
Approximate floating point reciprocal square root

**-M[no]fpmisalign**
Instructs the compiler to allow (not allow) vector arithmetic instructions with memory operands that are not aligned on 16-byte boundaries. The default is `-Mnofpmisalign` on all processors.

**-M[no]fprelaxed[=option]**
Instructs the compiler to use [not use] relaxed precision in the calculation of some intrinsic functions. Can result in improved performance at the expense of numerical accuracy. The possible values for option are:

**div**
Perform divide using relaxed precision.

**intrinsic**
Enables use of relaxed precision intrinsics.

**noorder**
Do not allow expression reordering or factoring.

**order**
Allow expression reordering, including factoring.

**recip**
Perform reciprocal using relaxed precision.

**rsqrt**
Perform reciprocal square root (1/sqrt) using relaxed precision.

**sqrt**
Perform square root with relaxed precision.

With no options, `-Mfprelaxed` generates relaxed precision code for those operations that generate a significant performance improvement, depending on the target processor. The default is `-Mnofprelaxed` which instructs the compiler to not use relaxed precision in the calculation of intrinsic functions.

**-Mi4**
(Fortran only) instructs the compiler to treat INTEGER variables as INTEGER*4.

**-Mlre[=array \| assoc \| noassoc]**
Enables loop-carried redundancy elimination, an optimization that can reduce the number of arithmetic operations and memory references in loops. The available suboptions are:

**assoc**
allow expression re-association. Specifying this suboption can increase opportunities for loop-carried redundancy elimination but may alter numerical results.

**noassoc**
disallow expression re-association.

**-Mnolre**
Disable loop-carried redundancy elimination.

**-Mnoframe**
Eliminate operations that set up a true stack frame pointer for every function. With this option enabled, you cannot perform a traceback on the generated code and you cannot access local variables.

**-Mnoi4**
(Fortran only) instructs the compiler to treat INTEGER variables as INTEGER*2.

**-Mpre**
Enables partial redundancy elimination.

**-Mprefetch[=option [,option...]]**
enables generation of prefetch instructions on processors where they are supported. Possible values for option include:

**d:m**
set the fetch-ahead distance for prefetch instructions to m cache lines.

**n:p**
set the maximum number of prefetch instructions to generate for a given loop to p.

**nta**
use the prefetch instruction.

**plain**
use the prefetch instruction (default).

**t0** use the prefetcht0 instruction.

**w** use the AMD-specific prefetchw instruction.

**-Mnoprefetch**
Disables generation of prefetch instructions.

**-M[no]propcond**
Enables or disables constant propagation from assertions derived from equality conditionals. The default is enabled.

**-Mr8**
(Fortran only) The compiler promotes REAL variables and constants to DOUBLE PRECISION variables and constants, respectively. DOUBLE PRECISION elements are 8 bytes in length.

**-Mnor8**
(Fortran only) The compiler does not promote REAL variables and constants to DOUBLE PRECISION. REAL variables will be single precision (4 bytes in length).

**-Mr8intrinsics**
(Fortran only) The compiler treats the intrinsics CMPLX and REAL as DCMPLX and DBLE, respectively.

**-Mnor8intrinsics**
(Fortran only) The compiler does not promote the intrinsics CMPLX and REAL to DCMPLX and DBLE, respectively.

**-Msafeptr[=option[,option,...]]**
(C++ and C only) instructs the C++ or C compiler to override data dependencies between pointers of a given storage class. Possible values of option include:

**all**
assume all pointers and arrays are independent and safe for aggressive optimizations, and in particular that no pointers or arrays overlap or conflict with each other.

**arg**
instructs the compiler to treat arrays and pointers with the same copyin and copyout semantics as Fortran dummy arguments.

**global**
instructs the compiler that global or external pointers and arrays do not overlap or conflict with each other and are independent.

**local / auto**
instructs the compiler that local pointers and arrays do not overlap or conflict with each other and are independent.

**static**
instructs the compiler that static pointers and arrays do not overlap or conflict with each other and are independent.

**-M[no]target_temps**
instructs the compiler to enable [disable] using temporaries when passing an array for a callee assumed-shape variable with the target attribute.

**-Munroll[=option [,option...]]**
invokes the loop unroller to execute multiple instances of the loop during each iteration. This also sets the optimization level to 2 if the level is set to less than 2, or if no -0 or -g options are supplied. The option is one of the following:

**c:m**
instructs the compiler to completely unroll loops with a constant loop count less than or equal to m, a supplied constant. If this value is not supplied, the m count is set to 1.

**m:<n>**
instructs the compiler to unroll multi-block loops n times. This option is useful for loops that have conditional statements. If n is not supplied, then the default value is 1. The default setting is not to enable -Munroll=m.

**n:<n>**
instructs the compiler to unroll single-block loops n times, a loop that is not completely unrolled, or has a non-constant loop count. If n is not supplied, the unroller computes the number of times a candidate loop is unrolled.

**-Mnounroll**
instructs the compiler not to unroll loops.

**-M[no]vect[=option [,option,...]]**
enable [disable] the code vectorizer, where option is one of the following:

**[no]altcode**
Enable [disable] generation of alternate code (altcode) for vectorized loops when appropriate. For each vectorized loop the compiler decides whether to generate altcode and what type or types to generate, which may be any or all of: altcode without iteration peeling, altcode with non-temporal stores and other data cache optimizations, and altcode based on array alignments calculated dynamically at runtime. The compiler also determines suitable loop count and array alignment conditionals for executing the al tcode. This option is enabled by default.

**[no]assoc**
Enable [disable] certain associativity conversions that can change the results of a computation due to roundoff error. A typical optimization is to change an arithmetic operation to an arithmetic operation that is mathematically correct, but can be computationally different, due to round-off error.

**cachesize:n**
Instructs the vectorizer, when performing cache tiling optimizations, to assume a cache size of n. The default is set per processor type, either using the `-tp` switch or auto-detected from the host computer.

**[no]fuse**
Enable [disable] automatic loop fusion by the vectorizer.

**[no]gather**
Enable [disable] vectorization of loops containing indirect array references, such as this one:

```
sum = 0.d0
do k=d(j),d(j+1)-1
     sum = sum + a(k)*b(c(k))
enddo
```

The default is gather.

**[no]idiom**
Enable [disable] idiom recognition by the vectorizer.

**levels:n**
Maximum nest level of loops to optimize

**nocond**
Disable vectorization of loops with conditionals.

**[no]partial**
Enable [disable] partial loop vectorization through innermost loop distribution.

**prefetch**
Instructs the vectorizer to search for vectorizable loops and, wherever possible, make use of prefetch instructions.

**[no]short**
Enable [disable] short vector operations. `-Mvect=short` enables generation of packed SIMD instructions for short vector operations that arise from scalar code outside of loops or within the body of a loop iteration.

**[no]simd[:{128|256|512}]**
Enable [disable] vectorization using SIMD instructions and data, either 128 bits, 256 bits or 512 bits wide, on processors where there is a choice.

**[no]simdresidual**
Enable [disable] vectorization using SIMD instructions of the residual iterations of a vectorized loop.

**[no]sizelimit:n**
Instructs the vectorizer to generate vector code for all loops where possible regardless of the number of statements in the loop. This overrides a heuristic in the vectorizer that ordinarily prevents vectorization of loops with a number of statements that exceeds a certain threshold. The default is nosizelimit.

**[no]uniform**
Instructs the vectorizer to perform the same optimizations in the vectorized and residual loops.

**Note:** This option may affect the performance of the residual loop.

**-Mnovintr**
instructs the compiler not to perform idiom recognition or introduce calls to hand-optimized vector functions.

## 3.4.7. Miscellaneous Controls

This section describes the -M<nvflag> options that do not easily fit into one of the other categories of -M<nvflag> options.

**Default:** Before looking at all the options, let's look at the defaults. For arguments that you do not specify, the default miscellaneous options are as follows:

| | | | |
|---|---|---|---|
| inform | nobounds | nolist | warn |

**Related options:** `-m`, `-S`, `-V`, `-v`

**Usage:** In the following example, the compiler includes Fortran source code with the assembly code.

```
$ nvfortran -Manno -S myprog.f
```

In the following example, the assembler does not delete the assembly file `myprog.s` after the assembly pass.

```
$ nvfortran -Mkeepasm myprog.f
```

In the following example, the compiler displays information about inlined functions with fewer than approximately 20 source lines in the source file `myprog.f`.

```
$ nvfortran -Minfo=inline -Minline=20 myprog.f
```

In the following example, the compiler creates the listing file `myprog.lst`.

```
$ nvfortran -Mlist myprog.f
```

In the following example, array bounds checking is enabled.

```
$ nvfortran -Mbounds myprog.f
```

The following list provides the syntax for each miscellaneous -M<nvflag> option. Each option has a description and, if appropriate, a list of any related options.

**-Manno**
annotate the generated assembly code with source code. Implies `-Mkeepasm`.

**-M[no]bounds**
Enables [disables] array bounds checking.

- If an array is an assumed size array, the bounds checking only applies to the lower bound.
- If an array bounds violation occurs during execution, an error message describing the error is printed and the program terminates. The text of the error message includes the name of the array, the location where the error occurred (the source file and the line number in the source), and information about the out of bounds subscript (its value, its lower and upper bounds, and its dimension).

The following is a sample error message:

```
NVFTN-F-Subscript out of range for array a (a.f: 2)
subscript=3, lower bound=1, upper bound=2, dimension=2
```

**-Mbyteswapio**
swap byte-order from big-endian to little-endian or vice versa upon input/output of Fortran unformatted data files.

**-Mchkptr**
instructs the compiler to check for pointers that are dereferenced while initialized to NULL (Fortran only).

**-Mchkstk**
instructs the compiler to check the stack for available space in the prologue of a function and before the start of a parallel region. Prints a warning message and aborts the program gracefully if stack space is insufficient. This option is useful when many local and private variables are declared in an OpenMP program.

**-Mcpp[=option [,option,...]]**
run the cpp-like preprocessor without execution of any subsequent compilation steps. This option is useful for generating dependence information to be included in makefiles.

---

**Note:** Only one of the m, md, mm or mmd options can be present; if multiple of these options are listed, the last one listed is accepted and the others are ignored.

---

The option is one or more of the following:

**m** print makefile dependencies to stdout.

**md** print makefile dependencies to `filename.d`, where filename is the root name of the input file being processed, ignoring system include files.

**mm** print makefile dependencies to stdout, ignoring system include files.

**mmd**
print makefile dependencies to `filename.d`, where filename is the root name of the input file being processed, ignoring system include files.

**[no]comment**
do [do not] retain comments in output.

**[suffix:]<suff>**
use <suff> as the suffix of the output file containing makefile dependencies.

**-Mgccbug[s]**
instructs the compiler to match the behavior of certain gcc bugs.

**-Miface[=option]**
adjusts the calling conventions for Fortran, where option is one of the following:

**cref**
uses CREF calling conventions, no trailing underscores.

**mixed_str_len_arg**
places the lengths of character arguments immediately after their corresponding argument. Has affect only with the CREF calling convention.

**nomixed_str_len_arg**
places the lengths of character arguments at the end of the argument list. Has affect only with the CREF calling convention.

**-Minfo[=option [,option,...]]**
instructs the compiler to produce information on standard error, where option is one of the following:

**all**
instructs the compiler to produce all available `-Minfo` information. Implies a number of suboptions:

```
-Minfo=accel,inline,ipa,loop,lre,mp,opt,par,vect,stdpar
```

**accel**
instructs the compiler to enable accelerator information.

**ftn**
instructs the compiler to enable Fortran-specific information.

**inline**
instructs the compiler to display information about extracted or inlined functions. This option is not useful without either the `-Mextract` or `-Minline` option.

**intensity**
instructs the compiler to provide informational messages about the intensity of the loop. Specify <n> to get messages on nested loops.

- For floating point loops, intensity is defined as the number of floating point operations divided by the number of floating point loads and stores.
- For integer loops, the loop intensity is defined as the total number of integer arithmetic operations, which may include updates of loop counts and addresses, divided by the total number of integer loads and stores.
- By default, the messages just apply to innermost loops.

**loop**
instructs the compiler to display information about loops, such as information on vectorization.

**lre**
instructs the compiler to enable LRE, loop-carried redundancy elimination, information.

**mp** instructs the compiler to display information about parallelization.

**opt**
instructs the compiler to display information about optimization.

**par**
instructs the compiler to enable parallelizer information.

**stdpar**
instructs the compiler to emit information about parallelization of C++ parallel algorithms and Fortran DO CONCURRENT loops.

**time**
instructs the compiler to display compilation statistics.

**unroll**
instructs the compiler to display information about loop unrolling.

**vect**
instructs the compiler to enable vectorizer information.

**-Minform=level**
instructs the compiler to display error messages at the specified and higher levels, where `level` is one of the following:

**fatal**
instructs the compiler to display fatal error messages.

**[no]file**
instructs the compiler to print or not print source file names as they are compiled. The default is to print the names: `-Minform=file`.

**inform**
instructs the compiler to display all error messages (inform, warn, severe and fatal).

**severe**
instructs the compiler to display severe and fatal error messages.

**warn**
instructs the compiler to display warning, severe and fatal error messages.

**-Mkeepasm**
instructs the compiler to keep the assembly file as compilation continues. Normally, the assembler deletes this file when it is finished. The assembly file has the same filename as the source file, but with a .s extension.

**-Mlist**
instructs the compiler to create a listing file. The listing file is `filename.lst`, where the name of the source file is `filename.f`.

**-Mnames={lowercase|uppercase}**
specifies the case for the names of Fortran externals.

- lowercase - Use lowercase for Fortran externals.
- uppercase - Use uppercase for Fortran externals.

**-Mneginfo[=option[,option,...]]**
instructs the compiler to produce information on standard error, where option is one of the following:

**all**
instructs the compiler to produce all available information on why various optimizations are not performed.

**accel**
instructs the compiler to enable accelerator information.

**concur**
instructs the compiler to produce all available information on why loops are not automatically parallelized. In particular, if a loop is not parallelized due to potential data dependence, the variable(s) that cause the potential dependence are listed in the messages that you see when using the option `-Mneginfo`.

**ftn**
instructs the compiler to enable Fortran-specific information.

**inline**
instructs the compiler to display information about extracted or inlined functions. This option is not useful without either the `-Mextract` or `-Minline` option.

**loop**
instructs the compiler to display information about loops, such as information on vectorization.

**lre**
instructs the compiler to enable LRE, loop-carried redundancy elimination, information.

**mp** instructs the compiler to display information about parallelization.

**opt**
instructs the compiler to display information about optimization.

**par**
instructs the compiler to enable parallelizer information.

**vect**
instructs the compiler to enable vectorizer information.

**-Mnolist**
the compiler does not create a listing file. This is the default.

**-Mnorpath**
Do not add -rpath to the link line.

**-Mnvpl[=option, [option, ...]]**
instruct the compiler to link in the NVIDIA Performance Libraries (NVPL) into the application. Use this option without sub-options to link against all libraries in the NVPL, or with sub-options to link only against those NVPL libraries specified. To use the NVPL ScaLAPACK library, use `-Mscalapack -Mnvpl`. Refer to the section on `-Mscalapack` for more information about this option.

---

**Note:** The NVPL Libraries are only available for Arm CPUs. For more information about NVPL, please visit https://docs.nvidia.com/nvpl/

---

Valid options for this flag are:

**blas**
link in the NVPL BLAS library.

**fft**
link in the NVPL FFT library.

**lapack**
link in the NVPL LAPACK library. This option will also cause the NVPL BLAS library to be linked in, as BLAS is a dependency of LAPACK.

**rand**
link in the NVPL RAND library.

**sparse**
link in the NVPL Sparse library.

**tensor**
link in the NVPL Tensor library.

**-Mpreprocess**
instruct the compiler to perform cpp-like preprocessing on assembly and Fortran input source files.

**-Mwritable_strings**
stores string constants in the writable data segment.

---

**Note:** Options `-Xs` and `-Xst` include -Mwritable_strings.

---

**-Mscalapack**
instruct the compiler to link in the ScaLAPACK library. ScaLAPACK is a library of high-performance linear algebra routines for parallel distributed memory machines, which uses MPI as the underlying communication mechanism. If `-Mnvpl` is also specified on the command line, then this flag will link in the NVPL ScaLAPACK library into the application. Otherwise, the Netlib ScaLAPACK library will be used.

# Chapter 4. C++ Name Mangling

Name mangling transforms the names of entities so that the names include information on aspects of the entity's type and fully qualified name. This ability is necessary since the intermediate language into which a program is translated contains fewer and simpler name spaces than there are in the C++ language; specifically:

- Overloaded function names are not allowed in the intermediate language.
- Classes have their own scopes in C++, but not in the generated intermediate language. For example, an entity x from inside a class must not conflict with an entity x from the file scope.
- External names in the object code form a completely flat name space. The names of entities with external linkage must be projected onto that name space so that they do not conflict with one another. A function f from a class A, for example, must not have the same external name as a function f from class B.
- Some names are not names in the conventional sense of the word, they're not strings of alphanumeric characters, for example: operator=.

There are two main problems here:

1. Generating external names that will not clash.
2. Generating alphanumeric names for entities with strange names in C++.

Name mangling solves these problems by generating external names that will not clash, and alphanumeric names for entities with strange names in C++. It also solves the problem of generating hidden names for some behind-the-scenes language support in such a way that they match up across separate compilations.

You see mangled names if you view files that are translated by NVC++ or NVC, and you do not use tools that demangle the C++ names. Intermediate files that use mangled names include the assembly and object files created by the NVC++ command. To view demangled names, use the tool `nvdecode`, which takes input from stdin. `nvdecode` demangles NVC++ names.

```
prompt> nvdecode
_ZN1A1gEf
A::g(float)
```

The name mangling algorithm for the NVC++ compiler is IA-64 ABI compliant and is described at http://mentorembedded.github.io/cxx-abi. Refer to this document for a complete description of the name mangling algorithm.

# Chapter 5. Pre-defined Compiler Macros

The HPC compilers will pre-define certain compiler macros. The macros are defined as follows:

```
#define __NVCOMPILER_MAJOR__ 25
#define __NVCOMPILER_MINOR__ 5
#define __NVCOMPILER_PATCHLEVEL__ 0
#define __NVCOMPILER_CLANG_SSE_INTRINSICS_VERSION__ 60000
#define __NVCOMPILER 1
#define __NVCOMPILER_LLVM__ 1
```

# Chapter 6. Runtime Environment

This section describes details related to compiler code generation, including register conventions and calling conventions for x86-64 and OpenPOWER processor-based systems. It addresses these conventions for processors running Linux operating systems.

**Note:** In this section we sometimes refer to word, halfword, and double word. The equivalent byte information is word (4 byte), halfword (2 byte), and double word (8 byte).

## 6.1. Linux Programming Model

This section defines compiler and assembly language conventions for the use of certain aspects of an x86-64 or OpenPOWER processor running a Linux operating system. These standards must be followed to guarantee that compilers, application programs, and operating systems written by different people and organizations will work together. The conventions supported by the NVC ISO/ANSI C compiler implement the application binary interface (ABI) as defined in the *System V Application Binary Interface: AMD64 Architecture Processor Supplement* and the *OpenPOWER for Linux Supplement, Power Architecture 64-Bit ELF V2 ABI Specification* listed in the *Preface*.

### 6.1.1. x86-64 Function Calling Sequence

This section describes the standard function calling sequence, including the stack frame, register usage, and parameter passing.

**x86-64 Register Usage Conventions**

The following table defines the standard for register allocation. The x86-64 Architecture provides a variety of registers. All the general purpose registers, x87 registers, XMM registers, SSE registers and AVX registers are visible to all procedures in a running program.

Table 1: Table 14. x86-64 Register Allocation

| Type | Name | Purpose |
|---|---|---|
| General | %rax | 1st return register. When callee has a variable number of arguments, %al specifies the number of vector registers passed. |
| | %rbx | callee-saved; optional base pointer |
| | %rcx | pass 4th argument to functions |
| | %rdx | pass 3rd argument to functions; 2nd return register |
| | %rsp | stack pointer |
| | %rbp | callee-saved; optional stack frame pointer |
| | %rsi | pass 2nd argument to functions |
| | %rdi | pass 1st argument to functions |
| | %r8 | pass 5th argument to functions |
| | %r9 | pass 6th argument to functions |
| | %r10 | temporary register; pass a function's static chain pointer |
| | %r11 | temporary register |
| | %r12-r15 | callee-saved registers |
| XMM | %xmm0-%xmm1 | pass and return floating point arguments |
| | %xmm2-%xmm7 | pass floating point arguments |
| | %xmm8-%xmm15 | temporary registers |
| x87 | %st(0) | temporary register; return long double arguments |
| | %st(1) | temporary register; return long double arguments |
| | %st(2) - %st(7) | temporary registers |

### x86-64 Stack Frame Organization

In addition to the registers, each function has a frame on the run-time stack. This stack grows downward from high addresses. *Table 15* shows the stack frame organization.

Table 2: Table 15. Standard Stack Frame

| Position | Contents | Frame |
|---|---|---|
| 8n+16 (%rbp) | argument eightbyte n ... | previous |
| 16 (%rbp) | argument eightbyte 0 | |
| 8 (%rbp) | return address | current |
| 0 (%rbp) | caller's %rbp | current |
| -8 (%rbp) | unspecified ... | |
| 0 (%rsp) | variable size | |
| -128 (%rsp) | red zone | |

**x86-64 Stack Usage Conventions**

Key points concerning the stack frame:

- The end of the input argument area is aligned on a 16-byte boundary.
- The 128-byte area beyond the location of %rsp is called the red zone and can be used for temporary local data storage. This area is not modified by signal or interrupt handlers.
- A call instruction pushes the address of the next instruction (the return address) onto the stack. The return instruction pops the address off the stack and effectively continues execution at the next instruction after the call instruction. A function must preserve non-volatile registers, a register whose contents must be preserved across subroutine calls. Additionally, the called function must remove the return address from the stack, leaving the stack pointer (%rsp) with the value it had before the call instruction was executed.

All registers on an x86-64 system are visible to both a calling and a called function. Registers %rbx, %rsp, %rbp, %r12, %r13, %r14, and %r15 are non-volatile across function calls. Therefore, a function must preserve these registers' values for its caller. Remaining registers are volatile (scratch) registers, that is a register whose contents need not be preserved across subroutine calls. If a calling function wants to preserve such a register value across a function call, it must save its value explicitly.

Registers are used extensively in the standard calling sequence. The first six integer and pointer arguments are passed in these registers (listed in order): %rdi, %rsi, %rdx, %rcx, %r8, %r9. The first eight floating point arguments are passed in the first eight XMM registers: %xmm0, %xmm1, ..., %xmm7. The registers %rax and %rdx are used to return integer and pointer values. The registers %xmm0 and %xmm1 are used to return floating point values.

Additional registers with assigned roles in the standard calling sequence:

**%rsp**
The stack pointer holds the limit of the current stack frame, which is the address of the stack's bottom-most, valid word. The stack must be 16-byte aligned.

**%rbp**
The frame pointer holds a base address for the current stack frame. Consequently, a function has registers pointing to both ends of its frame. Incoming arguments reside in the previous frame, referenced as positive offsets from %rbp, while local variables reside in the current frame, referenced as negative offsets from %rbp. A function must preserve this register value for its caller.

**RFLAGS**
The flags register contains the system flags, such as the direction flag and the carry flag. The

direction flag must be set to the “forward” (i.e., zero) direction before entry and upon exit from a function. Other user flags have no specified role in the standard calling sequence and are not preserved.

**Floating Point Control Word**

The control word contains the floating-point flags, such as the rounding mode and exception masking. This register is initialized at process initialization time and its value must be preserved.

Signals can interrupt processes. Functions called during signal handling have no unusual restriction on their use of registers. Moreover, if a signal handling function returns, the process resumes its original execution path with registers restored to their original values. Thus, programs and compilers may freely use all registers without danger of signal handlers changing their values.

**x86-64 Functions Returning Scalars or No Value**

- A function that returns an integral or pointer value places its result in the next available register of the sequence %rax, %rdx.
- A function that returns a floating point value that fits in the XMM registers returns this value in the next available XMM register of the sequence %xmm0, %xmm1.
- An X87 floating-point return value appears on the top of the floating point stack in %st(0) as an 80-bit X87 number. If this X87 return value is a complex number, the real part of the value is returned in %st(0) and the imaginary part in %st(1).
- A function that returns a value in memory also returns the address of this memory in %rax.
- Functions that return no value (also called procedures or void functions) put no particular value in any register.

**x86-64 Functions Returning Structures or Unions**

A function can use either registers or memory to return a structure or union. The size and type of the structure or union determine how it is returned. If a structure or union is larger than 16 bytes, it is returned in memory allocated by the caller.

To determine whether a 16-byte or smaller structure or union can be returned in one or more return registers, examine the first eight bytes of the structure or union. The type or types of the structure or union’s fields making up these eight bytes determine how these eight bytes will be returned. If the eight bytes contain at least one integral type, the eight bytes will be returned in %rax even if non-integral types are also present in the eight bytes. If the eight bytes only contain floating point types, these eight bytes will be returned in %xmm0.

If the structure or union is larger than eight bytes but smaller than 17 bytes, examine the type or types of the fields making up the second eight bytes of the structure or union. If these eight bytes contain at least one integral type, these eight bytes will be returned in %rdx even if non-integral types are also present in the eight bytes. If the eight bytes only contain floating point types, these eight bytes will be returned in %xmm1.

If a structure or union is returned in memory, the caller provides the space for the return value and passes its address to the function as a “hidden” first argument in %rdi. This address will also be returned in %rax.

**x86-64 Integral and Pointer Arguments**

Integral and pointer arguments are passed to a function using the next available register of the sequence %rdi, %rsi, %rdx, %rcx, %r8, %r9. After this list of registers has been exhausted, all remaining integral and pointer arguments are passed to the function via the stack.

**x86-64 Floating-Point Arguments**

Float and double arguments are passed to a function using the next available XMM register taken in the order from %xmm0 to %xmm7. After this list of registers has been exhausted, all remaining float and double arguments are passed to the function via the stack.

#### x86-64 Structure and Union Arguments

Structure and union arguments can be passed to a function in either registers or on the stack. The size and type of the structure or union determine how it is passed. If a structure or union is larger than 16 bytes, it is passed to the function in memory.

To determine whether a 16-byte or smaller structure or union can be passed to a function in one or two registers, examine the first eight bytes of the structure or union. The type or types of the structure or union's fields making up these eight bytes determine how these eight bytes will be passed. If the eight bytes contain at least one integral type, the eight bytes will be passed in the first available general purpose register of the sequence %rdi, %rsi, %rdx, %rcx, %r8, %r9 even if non-integral types are also present in the eight bytes. If the eight bytes only contain floating point types, these eight bytes will be passed in the first available XMM register of the sequence from %xmm0 to %xmm7.

If the structure or union is larger than eight bytes but smaller than 17 bytes, examine the type or types of the fields making up the second eight bytes of the structure or union. If the eight bytes contain at least one integral type, the eight bytes will be passed in the next available general purpose register of the sequence %rdi, %rsi, %rdx, %rcx, %r8, %r9 even if non-integral types are also present in the eight bytes. If these eight bytes only contain floating point types, these eight bytes will be passed in the next available XMM register of the sequence from %xmm0 to %xmm7.

If the first or second eight bytes of the structure or union cannot be passed in a register for some reason, the entire structure or union must be passed in memory.

#### x86-64 Passing Arguments on the Stack

If there are arguments left after every argument register has been allocated, the remaining arguments are passed to the function on the stack. The unassigned arguments are pushed on the stack in reverse order, with the last argument pushed first.

#### x86-64 Parameter Passing

*Table 16* shows the register allocation and stack frame offsets for the function declaration and call shown in the following example. Both table and example are adapted from System V Application Binary Interface: AMD64 Architecture Processor Supplement.

```
typedef struct {
    int a, b;
    double d;
    }
    structparam;
    structparam s;
    int e, f, g, h, i, j, k;
    float flt;
    double m, n;
    extern void func(int e, int f, structparam s, int g, int h,
    float flt, double m, double n, int i, int j, int k);
    void func2()
    {
    func(e, f, s, g, h, flt, m, n, i, j, k);
    }
```

Table 3: Table 16. Register Allocation for Example A-2

| General Purpose Registers | Floating Point Registers | Stack Frame Offset |
|---|---|---|
| %rdi: e | %xmm0: s.d | 0: j |
| %rsi: f | %xmm1: flt | 8: k |
| %rdx: s.a,s.b | %xmm2: m | |
| %rcx: g | %xmm3: n | |
| %r8: h | | |
| %r9: i | | |

#### x86-64 Implementing a Stack

In general, compilers and programmers must maintain a software stack. The stack pointer, register %rsp, is set by the operating system for the application when the program is started. The stack must grow downwards from high addresses.

A separate frame pointer enables calls to routines that change the stack pointer to allocate space on the stack at run-time (e.g. alloca). Some languages can also return values from a routine allocated on stack space below the original top-of-stack pointer. Such a routine prevents the calling function from using %rsp-relative addressing for values on the stack. If the compiler does not call routines that leave %rsp in an altered state when they return, a frame pointer is not needed and may not be used if the compiler option -Mnoframe is specified.

The stack must be kept aligned on 16-byte boundaries.

#### x86-64 Variable Length Parameter Lists

Parameter passing in registers can handle a variable number of parameters. The C language uses a special method to access variable-count parameters. The `stdarg.h` and `varargs.h` files define several functions to access these parameters. A C routine with variable parameters must use the va_start macro to set up a data structure before the parameters can be used. The `va_arg` macro must be used to access the successive parameters.

For calls that use `varargs` or `stdargs`, the register %rax acts as a hidden argument whose value is the number of XMM registers used in the call.

#### x86-64 C Parameter Conversion

In C, for a called prototyped function, the parameter type in the called function must match the argument type in the calling function. If the called function is not prototyped, the calling convention uses the types of the arguments but promotes char or short to int, and unsigned char or unsigned short to unsigned int and promotes float to double, unless you use the `-Msingle` option. For more information on the `-Msingle` option, refer to *-M Options by Category* .

#### x86-64 Calling Assembly Language Programs

The following example shows a C program calling an assembly-language routine `sum_3`.

#### C Program Calling an Assembly-language Routine

```
/* File: testmain.c */
#include <stdio.h>
int
main() {
 long l_para1 = 2;
```

(continues on next page)

(continued from previous page)

```
 float f_para2 = 1.0;
 double d_para3 = 0.5;
 float f_return;
 extern float sum_3(long para1, float para2, double para3);
 f_return = sum_3(l_para1, f_para2, d_para3);
 printf("Parameter one, type long = %ld\n", l_para1);
 printf("Parameter two, type float = %f\n", f_para2);
 printf("Parameter three, type double = %f\n", d_para3);
 printf("The sum after conversion = %f\n", f_return);
 return 0;
}
# File: sum_3.s
# Computes ( para1 + para2 ) + para3
    .text
    .align  16
    .globl  sum_3
sum_3:
    pushq   %rbp
    movq    %rsp, %rbp
    cvtsi2ssq %rdi, %xmm2
    addss   %xmm0, %xmm2
    cvtss2sd %xmm2,%xmm2
    addsd %xmm1, %xmm2
    cvtsd2ss %xmm2, %xmm2
    movaps  %xmm2, %xmm0
    popq    %rbp
    ret
    .type   sum_3, @function
    .size   sum_3,.-sum_3
```

### 6.1.2. OpenPOWER Function Calling Sequence

**OpenPOWER Register Usage Conventions**

The following table defines the standard for register allocation. The OpenPOWER Architecture provides a variety of registers. All the general purpose registers, vector scalar registers, and vector registers are visible to all procedures in a running program.

In the 64-bit OpenPOWER Architecture, there are always 32 general-purpose registers, each 64 bits wide. Throughout this document the symbol rN is used, where N is a register number, to refer to general-purpose register N.

Table 4: Table 17. OpenPOWER Register Allocation

| Type | Name | Preservation Rules | Purpose |
|---|---|---|---|
| General | r0 | Volatile | Optional use in function linkage. Used in function prologues. |
| | r1 | Nonvolatile | Stack frame pointer. |
| | r2 | Nonvolatile[1] | TOC pointer. |
| | r3–r10 | Volatile | Parameter and return values. |
| | r11 | Volatile | Optional use in function linkage. Used as an environment pointer in languages that require environment pointers. |
| | r12 | Volatile | Optional use in function linkage. Function entry address at the global entry point. |
| | r13 | Reserved | Thread pointer. |
| | r14–r31[2] | Nonvolatile | Local variables. |
| Floating-point | f0 | Volatile | Local variables. |
| | f1–f13 | Volatile | Used for parameter passing and return values of binary float types. |
| | f14–f31 | Nonvolatile | Local variables. |
| Vector | v0–v1 | Volatile | Local variables. |
| | v2–v13 | Volatile | Used for parameter passing and return values. |
| | v14–v19 | Volatile | Local variables. |
| | v20–v31 | Nonvolatile | Local variables. |

[1] Register r2 is nonvolatile with respect to calls between functions in the same compilation unit. It is saved and restored by code inserted by the linker resolving a call to an external function.

[2] If a function needs a frame pointer, assigning r31 to the role of the frame pointer is recommended.

In OpenPOWER-compliant processors, floating-point and vector functions are implemented using a unified vector-scalar model. As shown in *Figure 3* and *Figure 4*, there are 64 vector-scalar registers; each is 128 bits wide.

The vector-scalar registers can be addressed with vector-scalar instructions, for vector and scalar processing of all 64 registers, or with the “classic” Power floating-point instructions to refer to a 32-register subset of 64 bits per register. They can also be addressed with VMX instructions to refer to a 32-register subset of 128-bit wide registers.

The classic floating-point repertoire consists of 32 floating-point registers, each 64 bits wide, and an associated special-purpose register to provide floating-point status and control. Throughout this document, the symbol fN is used, where N is a register number, to refer to floating-point register N.

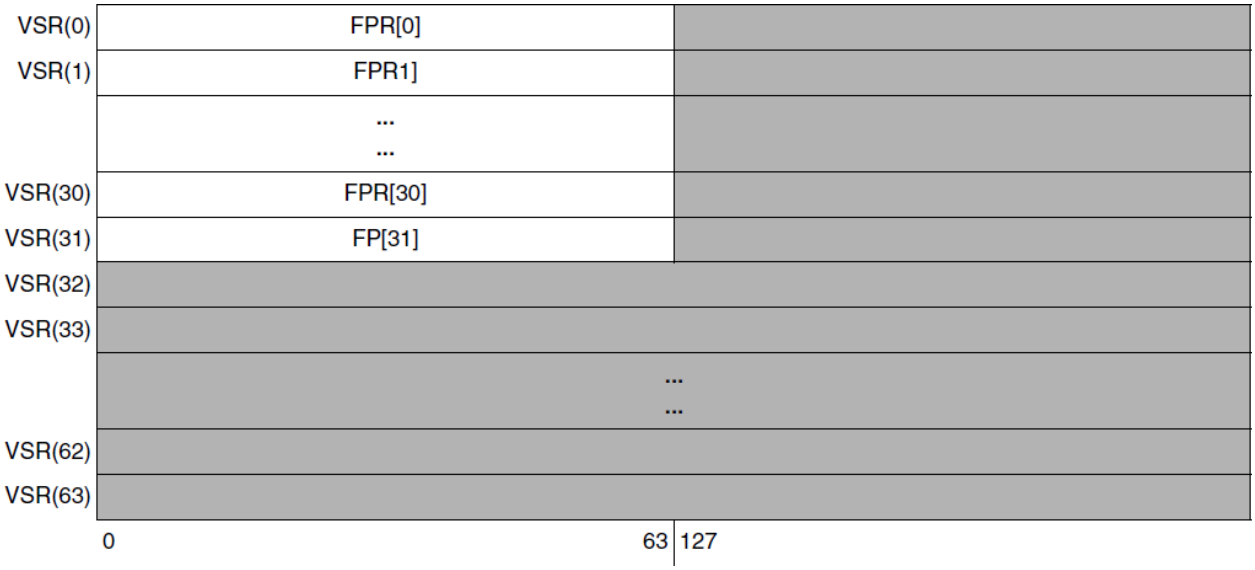


Fig. 1: Figure 3. Floating-point Registers as Part of Vector Scalar Registers

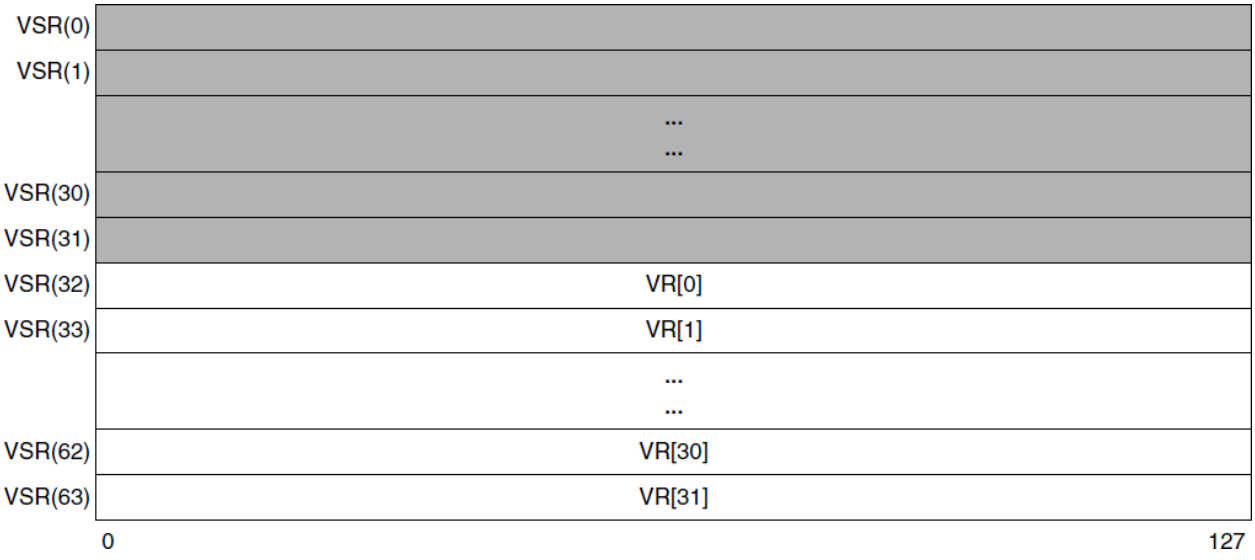


Fig. 2: Figure 4. Vector Registers as Part of Vector Scalar Registers

For the purpose of function calls, the right half of VSX registers, corresponding to the classic floating-point registers (that is, vsr0–vsr31), is volatile.

Single-precision and double-precision shall be passed in the floating-point registers. Single-precision decimal floating-point shall occupy the lower half of a floating-point register. When a floating-point register is skipped during input parameter allocation, words in the corresponding GPR or memory doubleword in the parameter list are not skipped.

The OpenPOWER vector-category instruction repertoire provides the ability to reference 32 vector registers, each 128 bits wide, of the vector-scalar register file, and a special-purpose register VSCR. Throughout this document, the symbol vN is used, where N is a register number, to refer to vector register N.

Parameters in the long double format with a pair of two double-precision floating-point values shall be passed in two successive floating-point registers.

If only one value can be passed in a floating-point register, the second parameter will be passed in a GPR or in memory in accordance with the parameter passing rules for structure aggregates.

**OpenPOWER Stack Frame Organization**

**OpenPOWER Stack Usage Conventions**

- The stack shall be quadword aligned.
- The minimum stack frame size shall be 32 bytes. A minimum stack frame consists of the first 4 doublewords (back-chain doubleword, CR save word and reserved word, LR save doubleword, and TOC pointer doubleword), with padding to meet the 16-byte alignment requirement.
- There is no maximum stack frame size defined.
- Padding shall be added to the Local Variable Space of the stack frame to maintain the defined stack frame alignment.
- The stack pointer, r1, shall always point to the lowest address doubleword of the most recently allocated stack frame.
- The stack shall start at high addresses and grow downward toward lower addresses.
- The lowest address doubleword (the back-chain word in *Figure 5*) shall point to the previously allocated stack frame when a back chain is present. As an exception, the first stack frame shall have a value of 0 (NULL).
- If required, the stack pointer shall be decremented in the called function's prologue and restored in the called function's epilogue.
- The stack pointer shall be updated atomically so that, at all times, it points to a valid back-chain doubleword if a back chain is maintained.
- Before a function calls any other functions, it shall save the value of the LR register into the LR save doubleword of the caller's stack frame.

Back Chain Doubleword

When a back chain is not present, alternate information compatible with the ABI unwind framework to unwind a stack must be provided by the compiler, for all languages, regardless of language features. A compiler that does not provide such system-compatible unwind information must generate a back chain. All compilers shall generate back chain information by default, and default libraries shall contain a back chain.

CR Save Word

If a function changes the value in any nonvolatile field of the condition register, it shall first save at least the value of those nonvolatile fields of the condition register, to restore before function exit. The

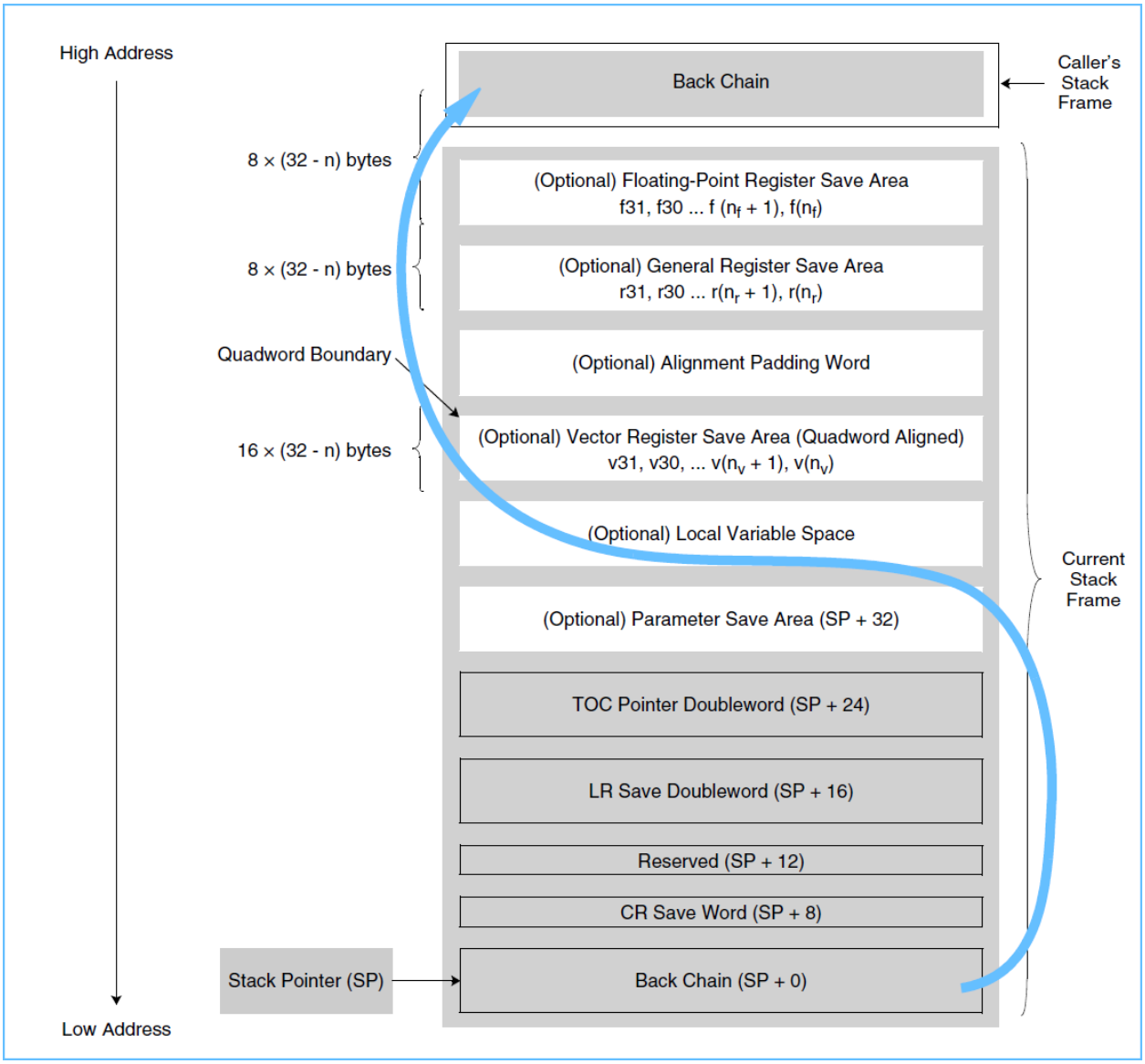


Fig. 3: Figure 5. Stack Frame Organization

caller frame CR Save Word may be used as the save location. This location in the current frame may be used as temporary storage, which is volatile over function calls.

Reserved Word

This word is reserved for system functions. Modifications of the value contained in this word are prohibited unless explicitly allowed by future ABI amendments.

LR Save Doubleword

If a function changes the value of the link register, it must first save the old value to restore before function exit. The caller frame LR Save Doubleword may be used as the save location. This location in the current frame may be used as temporary storage, which is volatile over a function call.

TOC Pointer Doubleword

If a function changes the value of the TOC pointer register, it shall first save it in the TOC pointer doubleword.

### OpenPOWER Optional Save Areas

This ABI provides a stack frame with a number of optional save areas. These areas are always present, but may be of size 0. This section indicates the relative position of these save areas in relation to each other and the primary elements of the stack frame.

Because the back-chain word of a stack frame must maintain quadword alignment, a reserved word is introduced above the CR save word to provide a quadword-aligned minimal stack frame and align the doublewords within the fixed stack frame portion at doubleword boundaries.

An optional alignment padding to a quadword-boundary element might be necessary above the Vector Register Save Area to provide 16-byte alignment, as shown in *Figure 5*.

Floating-Point Register Save Area

If a function changes the value in any nonvolatile floating-point register fN, it shall first save the value in fN in the Floating-Point Register Save Area and restore the register upon function exit.

The Floating-Point Register Save Area is always doubleword aligned. The size of the Floating-Point Register Save Area depends upon the number of floating-point registers that must be saved. If no floating-point registers are to be saved, the Floating-Point Register Save Area has a zero size.

General-Purpose Register Save Area

If a function changes the value in any nonvolatile general-purpose register rN, it shall first save the value in rN in the General-Purpose Register Save Area and restore the register upon function exit.

The General-Purpose Register Save Area is always doubleword aligned. The size of the General-Purpose Register Save Area depends upon the number of general registers that must be saved. If no general-purpose registers are to be saved, the General-Purpose Register Save Area has a zero size.

Vector Register Save Area

If a function changes the value in any nonvolatile vector register vN, it shall first save the value in vN in the Vector Register Save Area and restore the register upon function exit.

The Vector Register Save Area is always quadword aligned. If necessary to ensure suitable alignment of the vector save area, a padding doubleword may be introduced between the vector register and General- Purpose Register Save Areas, and/or the Local Variable Space may be expanded to the next quadword boundary. The size of the Vector Register Save Area depends upon the number of vector registers that must be saved. It ranges from 0 bytes to a maximum of 192 bytes (12 × 16). If no vector registers are to be saved, the Vector Register Save Area has a zero size.

Local Variable Space

The Local Variable Space is used for allocation of local variables. The Local Variable Space is located immediately above the Parameter Save Area, at a higher address. There is no restriction on the size of this area.

---

**Note:** Sometimes a register spill area is needed. It is typically positioned above the Local Variable Space.

---

The Local Variable Space also contains any parameters that need to be assigned a memory address when the function's parameter list does not require a save area to be allocated by the caller.

Parameter Save Area

The Parameter Save Area shall be allocated by the caller for function calls unless a prototype is provided for the callee indicating that all parameters can be passed in registers. (This requires a Parameter Save Area to be created for functions where the number and type of parameters exceeds the registers available for parameter passing in registers, for those functions where the prototype contains an ellipsis to indicate a variadic function, and functions are declared without prototype.)

When the caller allocates the Parameter Save Area, it will always be automatically quadword aligned because it must always start at SP + 32. It shall be at least 8 doublewords in length. If a function needs to pass more than 8 doublewords of arguments, the Parameter Save Area shall be large enough to spill all register-based parameters and to contain the arguments that the caller stores in it.

The calling function cannot expect that the contents of this save area are valid when returning from the callee.

The Parameter Save Area, which is located at a fixed offset of 32 bytes from the stack pointer, is reserved in each stack frame for use as an argument list when an in-memory argument list is required. For example, a Parameter Save Area must be allocated by the caller when calling functions with the following characteristics:

- Prototyped functions where the parameters cannot be contained in the parameter registers
- Prototyped functions with variadic arguments
- Functions without a suitable declaration available to the caller to determine the called function's characteristics (for example, functions in C without a prototype in scope).

Under these circumstances, a minimum of 8 doublewords are always reserved. The size of this area must be sufficient to hold the longest argument list being passed by the function that owns the stack frame. Although not all arguments for a particular call are located in storage, when an in-memory parameter list is required, consider the parameters to be forming a list in this area. Each argument occupies one or more doublewords.

More arguments might be passed than can be stored in the parameter registers. In that case, the remaining arguments are stored in the Parameter Save Area. The values passed on the stack are identical to the values placed in registers. Therefore, the stack contains register images for the values that are not placed into registers.

This ABI uses a simple va_list type for variable lists to point to the memory location of the next parameter. Therefore, regardless of type, variable arguments must always be in the same location so that they can be found at runtime. The first 8 doublewords are located in general registers r3–r10. Any additional doublewords are located in the stack Parameter Save Area. Alignment requirements such as those for vector types may require the va_list pointer to first be aligned before accessing a value.

Follow these rules for parameter passing:

- Map each argument to enough doublewords in the Parameter Save Area to hold its value.

- Map single-precision floating-point values to the least-significant word in a single doubleword.
- Map double-precision floating-point values to a single doubleword.
- Map simple integer types (char, short, int, long, enum) to a single doubleword. Sign or zero extend values shorter than a doubleword to a doubleword based on whether the source data type is signed or unsigned.
- When 128-bit integer types are passed by value, map each to two consecutive GPRs, two consecutive doublewords, or a GPR and a doubleword. The required alignment of int128 data types is 16 bytes. Therefore, by-value parameters must be copied to a new location in the local variable area of the callee's stack frame before the address of the type can be provided (for example, using the address-of operator, or when the variable is to be passed by reference), when the incoming parameter is not aligned at a 16-byte boundary.
- Map long double to two consecutive doublewords. The required alignment of long double data types is 16 bytes. Therefore, by-value parameters must be copied to a new location in the local variable area of the callee's stack frame before the address of the type can be provided (for example, using the address-of operator, or when the variable is to be passed by reference), when the incoming parameter is not aligned at a 16-byte boundary.
- Map complex floating-point and complex integer types as if the argument was specified as separate real and imaginary parts.
- Map pointers to a single doubleword.
- Map vectors to a single quadword, quadword aligned. This might result in skipped doublewords in the Parameter Save Area.
- Map fixed-size aggregates and unions passed by value to as many doublewords of the Parameter Save Area as the value uses in memory. Align aggregates and unions as follows:
  - Aggregates that contain qualified floating-point or vector arguments are normally aligned at the alignment of their base type. For more information about qualified arguments, see *OpenPOWER Parameter Passing in Registers*.
  - Other aggregates are normally aligned in accordance with the aggregate's defined alignment.
  - The alignment will never be larger than the stack frame alignment (16 bytes).

  This might result in doublewords being skipped for alignment. When a doubleword in the Parameter Save Area (or its GPR copy) contains at least a portion of a structure, that doubleword must contain all other portions mapping to the same doubleword. (That is, a doubleword can either be completely valid, or completely invalid, but not partially valid and invalid, except in the last doubleword where invalid padding may be present.)
- Pad an aggregate or union smaller than one doubleword in size so that it is in the least-significant bits of the doubleword. Pad all others, if necessary, at their tail. Variable size aggregates or unions are passed by reference.
- Map other scalar values to the number of doublewords required by their size.
- Future data types that have an architecturally defined quadword-required alignment will be aligned at a quadword boundary.
- If the callee has a known prototype, arguments are converted to the type of the corresponding parameter when loaded to their parameter registers or when being mapped into the Parameter Save Area. For example, if a long is used as an argument to a float double parameter, the value is converted to double precision and mapped to a doubleword in the Parameter Save Area.

**OpenPOWER Protected Zone**

The 288 bytes below the stack pointer are available as volatile program storage that is not preserved across function calls. Interrupt handlers and any other functions that might run without an explicit call must take care to preserve a protected zone, also referred to as the red zone, of 512 bytes that consists of:

- The 288-byte volatile program storage region that is used to hold saved registers and local variables
- An additional 224 bytes below the volatile program storage region that is set aside as a volatile system storage region for system functions

If a function does not call other functions and does not need more stack space than is available in the volatile program storage region (that is, 288 bytes), it does not need to have a stack frame. The 224-byte volatile system storage region is not available to compilers for allocation to saved registers and local variables.

### OpenPOWER Parameter Passing in Registers

For the OpenPOWER Architecture, it is more efficient to pass arguments to functions in registers rather than through memory. For more information about passing parameters through memory, see *Parameter Save Area* under *OpenPOWER Optional Save Areas*. For the OpenPOWER ABI, the following parameters can be passed in registers:

- Up to eight arguments can be passed in general-purpose registers r3–r10.
- Up to thirteen qualified floating-point arguments can be passed in floating-point registers f1–f13 or up to twelve in vector registers v2–v13.
- Up to thirteen single-precision or double-precision decimal floating-point arguments can be passed in floating-point registers f1–f13.
- Up to six quad-precision decimal floating-point arguments can be passed in even-odd floating-point register pairs f2–f13.
- Up to 12 qualified vector arguments can be passed in v2–v13.

A qualified floating-point argument corresponds to:

- A scalar floating-point data type
- Each member of a complex floating-point type
- A member of a homogeneous aggregate of multiple like data types passed in up to eight floating-point registers

  A homogeneous aggregate can consist of a variety of nested constructs including structures, unions, and array members, which shall be traversed to determine the types and number of members of the base floating-point type. (A complex floating-point data type is treated as if two separate scalar values of the base type were passed.)

  Homogeneous floating-point aggregates can have up to four long double members or eight members of floating-point types. (Unions are treated as their largest member. For homogeneous unions, different union alternatives may have different sizes, provided that all union members are homogeneous with respect to each other.) They are passed in floating-point registers if parameters of that type would be passed in floating-point registers. They are passed in vector registers if parameters of that type would be passed in vector registers. They are passed as if each member was specified as a separate parameter.

A qualified vector argument corresponds to:

- A vector data type

- A member of a homogeneous aggregate of multiple like data types passed in up to eight vector registers
- Any future type requiring 16-byte alignment (see *OpenPOWER Optional Save Areas*) or processed in vector registers

  A homogeneous aggregate can consist of a variety of nested constructs including structures, unions, and array members, which shall be traversed to determine the types and number of members of the base vector type. Homogeneous vector aggregates with up to eight members are passed in up to eight vector registers as if each member was specified as a separate parameter. (Unions are treated as their largest member. For homogeneous unions, different union alternatives may have different sizes, provided that all union members are homogeneous with respect to each other.)

**Note:** Floating-point and vector aggregates that contain padding words and integer fields with a width of 0 should not be treated as homogeneous aggregates.

A homogeneous aggregate is either a homogeneous floating-point aggregate or a homogeneous vector aggregate. This ABI does not specify homogeneous aggregates for integer types.

Long double numbers in are passed using two successive floating-point registers. A floating-point register might be skipped to allocate an even/odd register pair when necessary. When a floating-point register is skipped, no corresponding memory word is skipped in the natural home location; that is, the corresponding GPR or memory doubleword in the parameter list.

All other aggregates are passed in consecutive GPRs, in GPRs and in memory, or in memory.

When a parameter is passed in a floating-point or vector register, a number of GPRs are skipped, in allocation order, commensurate to the size of the corresponding in-memory representation of the passed argument's type.

Each parameter is allocated to at least one doubleword.

*Full doubleword rule:*

When a doubleword in the Parameter Save Area (or its GPR copy) contains at least a portion of a structure, that doubleword must contain all other portions mapping to the same doubleword. (That is, a doubleword can either be completely valid, or completely invalid, but not partially valid and invalid, except in the last doubleword where invalid padding may be present.)

Long Double

Long double parameters are passed as if they were a struct consisting of separate double parameters.

Long double parameters shall be considered as a distinct type for the determination of homogeneous aggregates.

If fewer arguments are needed, the unused registers defined previously will contain undefined values on entry to the called function.

If there are more arguments than registers or no function prototype is provided, a function must provide space for all arguments in its stack frame. When this happens, only the minimum storage needed to contain all arguments (including allocating space for parameters passed in registers) needs to be allocated in the stack frame.

General-purpose registers r3–r10 correspond to the allocation of parameters to the first 8 doublewords of the Parameter Save Area. Specifically, this requires a suitable number of general-purpose registers to be skipped to correspond to parameters passed in floating-point and vector registers.

If a parameter corresponds to an unnamed parameter that corresponds to the ellipsis, a caller shall promote float values to double. If a parameter corresponds to an unnamed parameter that corresponds to the ellipsis, the parameter shall be passed in a GPR or in the Parameter Save Area.

If no function prototype is available, the caller shall promote float values to double and pass floating-point parameters in both available floating-point registers and in the Parameter Save Area. If no function prototype is available, the caller shall pass vector parameters in both available vector registers and in the Parameter Save Area. (If the callee expects a float parameter, the result will be incorrect.)

It is the callee's responsibility to allocate storage for the stored data in the local variable area. When the callee's parameter list indicates that the caller must allocate the Parameter Save Area (because at least one parameter must be passed in memory or an ellipsis is present in the prototype), the callee may use the preallocated Parameter Save Area to save incoming parameters.

**OpenPOWER Parameter Passing Register Selection Algorithm**

The following algorithm describes where arguments are passed for the C language. In this algorithm, arguments are assumed to be ordered from left (first argument) to right. The actual order of evaluation for arguments is unspecified.

- gr contains the number of the next available general-purpose register.
- fr contains the number of the next available floating-point register.
- vr contains the number of the next available vector register.

---

**Note:** The following types refer to the type of the argument as declared by the function prototype. The argument values are converted (if necessary) to the types of the prototype arguments before passing them to the called function.

---

If a prototype is not present, or it is a variable argument prototype and the argument is after the ellipsis, the type refers to the type of the data objects being passed to the called function.

- INITIALIZE: If the function return type requires a storage buffer, set gr = 4; else set gr = 3.

```
Set fr = 1
Set vr = 2
```

- SCAN: If there are no more arguments, terminate. Otherwise, allocate as follows based on the class of the function argument:

```
switch(class(argument))

integer:
pointer:

    if gr > 10
        goto mem_argument
    pass (GPR, gr, argument);
    gr++

    break;

aggregate:
    if (homogeneous(argument,float) and regs_needed(members(argument)) <= 8)
        n_fregs = n_fregs_for_type(member_type(argument,0))
        agg_size = members(argument * n_fregs
        reg_size = min(agg_size, 15-fr)
```

(continues on next page)

(continued from previous page)

```
        pass(FPR,fr,first_n_DW(argument,reg_size)
        fr += reg_size;
        gr += size_in_DW (first_n_DW(argument,reg_size))

        if remaining_members
            argument = after_n_DW(argument,reg_size))
            goto gpr_struct
    break;

    if (homogeneous(argument,vector) and members(argument) <= 8)
        use_vrs:
            agg_size = members(argument)
            reg_size = min(agg_size, 14-vr)
            if (gr&1 = 0) // align vector in memory
                gr++
            pass(VR,vr,first_n_elements(argument,reg_size);
            vr += reg_size
            gr += size_in_DW (first_n_elements(argument,reg_size)

            if remaining_members
                argument = after_n_elements(argument,reg_size))
                goto gpr_struct

    break;

    if gr > 10
        goto mem_argument

    size = size_in_DW(argument)

gpr_struct:
    reg_size = min(size, 11-gr)
    pass (GPR, gr, first_n_DW (argument, reg_size));
    gr += size_in_DW (first_n_DW (argument, reg_size))

    if remaining_members
        argument = after_n_DW(argument,reg_size))
        goto mem_argument

    break;

float:

// float is passed in one FPR.
// double is passed in one FPR.

    if (register_type_used (type (argument)) == vr)
        goto use_vr;
    if fr > 14
        goto mem_argument

    n_fregs = n_fregs_for_type(argument) // Assumes n_fregs_for_type == 2
                                         // for long double == 1 for float
                                         // or double
    pass(FPR,fr,argument)
    fr += n_fregs
```

(continues on next page)

(continued from previous page)

```
    gr += size_in_DW(argument)

    break;

vector:
    Use vr:
        if vr > 13
            goto mem_argument

        if (gr&1 = 0) // align vector in memory
            gr++

        pass(VR,vr,argument)
        vr ++
        gr += 2

        break;

next argument;

mem_argument:
    need_save_area = TRUE
    pass (stack, gr, argument)
    gr += size_in_DW(argument)

next argument;
```

All complex data types are handled as if two scalar values of the base type were passed as separate parameters.

If the callee takes the address of any of its parameters, values passed in registers are stored to memory. It is the callee's responsibility to allocate storage for the stored data in the local variable area. When the callee's parameter list indicates that the caller must allocate the Parameter Save Area (because at least one parameter must be passed in memory, or an ellipsis is present in the prototype), the callee may use the preallocated Parameter Save Area to save incoming parameters. (If an ellipsis is present, using the preallocated Parameter Save Area ensures that all arguments are contiguous.) If the compilation unit for the caller contains a function prototype, but the callee has a mismatching definition, this may result in the wrong values being stored.

---

**Note:** If the declaration of a function that is used by the caller does not match the definition for the called function, corruption of the caller's stack space can occur.

---

#### OpenPOWER Variable Argument Lists

C programs that are intended to be portable across different compilers and architectures must use the header file <stdarg.h> to deal with variable argument lists. This header file contains a set of macro definitions that define how to step through an argument list. The implementation of this header file may vary across different architectures, but the interface is the same.

C programs that do not use this header file for the variable argument list and assume that all the arguments are passed on the stack in increasing order on the stack are not portable, especially on architectures that pass some of the arguments in registers. The OpenPOWER Architecture is one of the architectures that passes some of the arguments in registers.

The parameter list may be zero length and is only allocated when parameters are spilled, when a function has unnamed parameters, or when no prototype is provided. When the Parameter Save Area is

allocated, the Parameter Save Area must be large enough to accommodate all parameters, including parameters passed in registers.

**OpenPOWER Return Values**

Functions that return a value shall place the result in the same registers as if the return value was the first named input argument to a function unless the return value is a nonhomogeneous aggregate larger than 2 doublewords or a homogeneous aggregate with more than eight registers. For a definition of homogeneous aggregates, see *OpenPOWER Parameter Passing in Registers*. (Homogeneous aggregates are arrays, structs, or unions of a homogeneous floating-point or vector type and of a known fixed size.) Therefore, long double functions are returned in f1:f2.

Homogeneous floating-point or vector aggregate return values that consist of up to eight registers with up to eight elements will be returned in floating-point or vector registers that correspond to the parameter registers that would be used if the return value type were the first input parameter to a function.

Aggregates that are not returned by value are returned in a storage buffer provided by the caller. The address is provided as a hidden first input argument in general-purpose register r3.

Functions that return values of the following types shall place the result in register r3 as signed or unsigned integers, as appropriate, and sign extended or zero extended to 64 bits where necessary:

- char
- enum
- short
- int
- long
- pointer to any type
- _Bool

## 6.1.3. Linux Fortran Supplement

Sections A2.4.1 through A2.4.4 of the ABI for Linux defines the Fortran supplement. The register usage conventions set forth in that document remain the same for Fortran.

**Fortran Fundamental Types**

Table 5: Table 18. Linux Fortran Fundamental Types

| Fortran Type | Size (bytes) | Alignment (bytes) |
|---|---|---|
| INTEGER | 4 | 4 |
| INTEGER*1 | 1 | 1 |
| INTEGER*2 | 2 | 2 |
| INTEGER*4 | 4 | 4 |
| INTEGER*8 | 8 | 8 |
| LOGICAL | 4 | 4 |
| LOGICAL*1 | 1 | 1 |
| LOGICAL*2 | 2 | 2 |
| LOGICAL*4 | 4 | 4 |
| LOGICAL*8 | 8 | 8 |
| BYTE | 1 | 1 |
| CHARACTER*n | n | 1 |
| REAL | 4 | 4 |
| REAL*4 | 4 | 4 |
| REAL*8 | 8 | 8 |
| DOUBLE PRECISION | 8 | 8 |
| COMPLEX | 8 | 4 |
| COMPLEX*8 | 8 | 4 |
| COMPLEX*16 | 16 | 8 |
| DOUBLE COMPLEX | 16 | 8 |

A logical constant is one of:

- .TRUE.
- .FALSE.

The logical constants .TRUE. and .FALSE. are defined to be the four-byte values -1 and 0 respectively. A logical expression is defined to be .TRUE. if its least significant bit is 1 and .FALSE. otherwise.

Note that the value of a character is not automatically NULL-terminated.

**Naming Conventions**

By default, all globally visible Fortran symbol names (subroutines, functions, common blocks) are converted to lower-case. In addition, an underscore is appended to Fortran global names to distinguish the Fortran name space from the C/C++ name space.

**Argument Passing and Return Conventions**

Arguments are passed by reference (i.e., the address of the argument is passed, rather than the argument itself). In contrast, C/C++ arguments are passed by value.

When passing an argument declared as Fortran type CHARACTER, an argument representing the

length of the CHARACTER argument is also passed to the function. This length argument is a four-byte integer passed by value, and is passed at the end of the parameter list following the other formal arguments. A length argument is passed for each CHARACTER argument; the length arguments are passed in the same order as their respective CHARACTER arguments.

A Fortran function, returning a value of type CHARACTER, adds two arguments to the beginning of its argument list. The first additional argument is the address of the area created by the caller for the return value; the second additional argument is the length of the return value. If a Fortran function is declared to return a character value of constant length, for example CHARACTER*4 FUNCTION CHF(), the second extra parameter representing the length of the return value must still be supplied.

On Linux86-64 systems a Fortran complex function returns its value in memory. The caller provides space for the return value and passes the address of this storage as if it were the first argument to the function. On OpenPOWER systems a Fortran complex function returns its value in the same manner as complex functions.

Alternate return specifiers of a Fortran function are not passed as arguments by the caller. The alternate return function passes the appropriate return value back to the caller in %rax on Linux86-64 and in r1 on OpenPOWER.

The handling of the following Fortran 90 features is implementation-defined: internal procedures, pointer arguments, assumed-shape arguments, functions returning arrays, and functions returning derived types.

**Inter-language Calling**

Inter-language calling between Fortran and C/C++ is possible if function/subroutine parameters and return values match types.

- If a C/C++ function returns a value, call it from Fortran as a function, otherwise, call it as a subroutine.
- If a Fortran function has type CHARACTER (or COMPLEX on Linux86-64), call it from C/C++ as a void function.
- If a Fortran subroutine has alternate returns, call it from C/C++ as a function returning int; the value of such a subroutine is the value of the integer expression specified in the alternate RETURN statement.
- If a Fortran subroutine does not contain alternate returns, call it from C/C++ as a void function.

Fortran 2003 also provides a mechanism to support interoperability with C. This mechanism inclues the ISO_C_BINDING intrinsic module, binding labels, and the BIND attribute.

*Table 19* provides the C/C++ data type corresponding to each Fortran data type.

Table 6: Table 19. Fortran and C/C++ Data Type Compatibility

| Fortran Type | C/C++ Type | Size (bytes) |
|---|---|---|
| CHARACTER*n x | char x[n] | n |
| REAL x | float x | 4 |
| REAL*4 x | float x | 4 |
| REAL*8 x | double x | 8 |
| DOUBLE PRECISION x | double x | 8 |
| INTEGER x | int x | 4 |
| INTEGER*1 x | signed char x | 1 |
| INTEGER*2 x | short x | 2 |
| INTEGER*4 x | int x | 4 |
| INTEGER*8 x | long x, or long long x | 8 |
| LOGICAL x | int x | 4 |
| LOGICAL*1 x | char x | 1 |
| LOGICAL*2 x | short x | 2 |
| LOGICAL*4 x | int x | 4 |
| LOGICAL*8 x | long x, or long long x | 8 |

Table 7: Table 20. Fortran and C/C++ Representation of the COMPLEX Type

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

**Note:** For C/C++, the `complex` type implies C99 or later.

**Arrays**

C/C++ arrays and Fortran arrays use different default initial array index values. By default, C/C++ arrays start at 0 and Fortran arrays start at 1. A Fortran array can be declared to start at zero.

Another difference between Fortran and C/C++ arrays is the storage method used. Fortran uses column-major order and C/C++ use row-major order. For one-dimensional arrays, this poses no problems. For two-dimensional arrays, where there are an equal number of rows and columns, row and column indexes can simply be reversed. Inter-language function mixing is not recommended for arrays other than single dimensional arrays and square two-dimensional arrays.

**Structures, Unions, Maps, and Derived Types**

Fields within Fortran structures and derived types, and multiple map declarations within a Fortran union, conform to the same alignment requirements used by C structures.

**Common Blocks**

A named Fortran common block can be represented in C/C++ by a structure whose members correspond to the members of the common block. The name of the structure in C/C++ must have the added underscore.

For example, the Fortran common block:

```
INTEGER I, J
COMPLEX C
DOUBLE COMPLEX CD
DOUBLE PRECISION D
COMMON /COM/ i, j, c, cd, d
```

is represented in C with the following equivalent:

```
extern struct {
    int i;
    int j;
    struct {float real, imag;} c;
    struct {double real, imag;} cd;
    double d;
} com_;
```

and in C++ with the following equivalent:

```
extern "C" struct {
    int i;
    int j;
    struct {float real, imag;} c;
    struct {double real, imag;} cd;
    double d;
} com_;
```

**Note:** The compiler-provided name of the BLANK COMMON block is implementation specific.

Calling Fortran COMPLEX and CHARACTER functions from C/C++ is not as straightforward as calling other types of Fortran functions. Additional arguments must be passed to the Fortran function by the C/C++ caller. A Fortran COMPLEX function returns its value in memory; the first argument passed to the function must contain the address of the storage for this value. A Fortran CHARACTER function adds two arguments to the beginning of its argument list. The following example of calling a Fortran CHARACTER function from C/C++ illustrates these caller-provided extra parameters:

```
CHARACTER*(*) FUNCTION CHF(C1, I)
CHARACTER*(*) C1
INTEGER I
END
```

```
extern void chf_();
char tmp[10];
char c1[9];
int i;
chf_(tmp, 10, c1, &i, 9);
```

The extra parameters tmp and 10 are supplied for the return value, while 9 is supplied as the length of c1.

# Chapter 7. C++ Dialect Supported

The NVC++ compiler accepts the C++ language of the ISO/IEC standards up to and including the 14882:2017 standard, plus substantially all GNU C++ extensions.

Command-line options provide full support of many C++ variants, including strict standard conformance. NVC++ provides the `--c++XY` command-line options to enable the user to specify the version of C++ accepted, where XY is one of `{17 \| 14 \| 11 \| 03}`. The C++ version accepted by default is determined by and matches that of the version of the GCC toolchain used for compilation.

## 7.1. C++17 Language Features Accepted

The NVC++ compiler includes support for the C++17 language standard. Enable this support by compiling with `--c++17` or `-std=c++17`.

Supported C++17 core language features are available on Linux systems using a GCC 7 or later toolchain.

The following C++17 language features are supported:

- Structured bindings
- Selection statements with initializers
- Compile-time conditional statements, a.k.a. `constexpr if`
- Fold expressions
- Inline variables
- Constexpr lambdas
- Lambda capture of *this by value
- Class template deduction
- Auto non-type template parameters
- Guaranteed copy elision

The NVC++ compiler installation does not include a C++ standard library, so support for C++17 additions to the standard library depends on the C++ library provided on your system. On Linux, GCC 7 is the first GCC release with significant C++17 support.

The following C++ library changes are supported when building against GCC 7 or later:

- std::string_view
- std::optional

- std::variant
- std::any
- Variable templates for metafunctions

The following C++ library changes are supported when building against GCC 9 or later:

- Parallel algorithms
- Filesystem support
- Polymorphic allocators and memory resources

# Chapter 8. x86-64 C++ and C MMX/SSE/AVX Intrinsics

An intrinsic is a function available in a given language whose implementation is handled specifically by the compiler. Typically, an intrinsic substitutes a sequence of automatically-generated instructions for the original function call. Since the compiler has an intimate knowledge of the intrinsic function, it can better integrate it and optimize it for the situation.

NVIDIA provides support for MMX and SSE/SSE2/SSE3/SSSE3/SSE4a/ABM/AVX intrinsics in C++ and C programs.

Intrinsics make the use of processor-specific enhancements easier because they provide a C++ and C language interface to assembly instructions. In doing so, the compiler manages things that the user would normally have to be concerned with, such as register names, register allocations, and memory locations of data.

This section contains these tables associated with inline intrinsics:

- A table of MMX inline intrinsics (mmintrin.h)
- A table of SSE inline intrinsics (xmmintrin.h)
- A table of SSE2 inline intrinsics (emmintrin.h)
- A table of SSE3 inline intrinsics (pmmintrin.h)
- A table of SSSE3 inline intrinsics (tmmintrin.h)
- A table of SSE4a inline intrinsics (ammintrin.h)
- A table of ABM inline intrinsics (intrin.h)
- A table of AVX inline intrinsics (immintrin.h)

## 8.1. Using Intrinsic functions

The definitions of the intrinsics are provided in the corresponding header files.

### 8.1.1. Required Header File

To call these intrinsic functions from a C/C++ source, you must include the corresponding header file – one of the following:

- For MMX, use `mmintrin.h`
- For SSE, use `xmmintrin.h`
- For SSE2, use `emmintrin.h`
- For SSE3, use `pmmintrin.h`
- For SSSE3 use `tmmintrin.h`
- For SSE4a use `ammintrin.h`
- For ABM use `intrin.h`
- For AVX use `intrin.h`

### 8.1.2. Intrinsic Data Types

The following table describes the data types that are defined for intrinsics:

| Data Types | Defined in | Description |
|---|---|---|
| __m64 | `mmintrin.h` \| For use with MMX intrinsics, this 64-bit data type stores one 64-bit or two 32-bit integer values. | |
| __m128 | `xmmintrin.h` \| For use with SSE intrinsics, this 128-bit data type, aligned on 16-byte boundaries, stores four single-precision floating point values. | |
| __m128d | `emmintrin.h` \| For use with SSE2/SSE3 intrinsics, this 128-bit data type, aligned on 16-byte boundaries, stores two double-precision floating point values. | |
| __ m128i | `emmintrin.h` \| For use with SSE2/SSE3 intrinsics, this 128-bit data type, aligned on 16-byte boundaries, stores two 64-bit integer values. | |
| __m256 | `immintrin.h` \| For use with AVX intrinsics, this 256-bit data type, aligned on 31-byte boundaries, stores eight single-precision floating point values. | |
| __m256d | `immintrin.h` \| For use with AVX intrinsics, this 256-bit data type, aligned on 32-byte boundaries, stores four double-precision floating point values. | |
| __m256i | `immintrin.h` \| For use with AVX intrinsics, this 256-bit data type, aligned on 16-byte boundaries, stores four 64-bit integer values. | |

### 8.1.3. Intrinsic Example

The MMX/SSE intrinsics include functions for initializing variables of the types defined in the preceding table. The following sample program, `example.c`, illustrates the use of the SSE intrinsics _mm_add_ps and _mm_set_ps.

```
#include<xmmintrin.h>
 int main(){
 __m128 A, B, result;
 A = _mm_set_ps(23.3, 43.7, 234.234, 98.746); /* initialize A */
 B = _mm_set_ps(15.4, 34.3, 4.1, 8.6); /* initialize B */
```

(continues on next page)

(continued from previous page)

```
result = _mm_add_ps(A, B);
return 0;
}
```

To compile this program, use the following command:

```
$ nvc example.c -o myprog
```

## 8.2. x86-64 MMX Intrinsics

NVC++ and NVC support a set of MMX Intrinsics which allow the use of the MMX instructions directly from C++ and C code, without writing the assembly instructions. The following table lists the MMX intrinsics supported.

**Note:** Intrinsics with a * are only available on 64-bit systems.

Table 1: Table 21. MMX Intrinsics (mmintrin.h)

| | | | |
|---|---|---|---|
| _mm_empty | _m_paddd | _m_psllw | _m_pand |
| _m_empty | _mm_add_si64 | _mm_slli_pi16 | _mm_andnot_si64 |
| _mm_cvtsi32_si64 | _mm_adds_pi8 | _m_psllwi | _m_pandn |
| _m_from_int | _m_paddsb | _mm_sll_pi32 | _mm_or_si64 |
| _mm_cvtsi64x_si64* | _mm_adds_pi16 | _m_pslld | _m_por |
| _mm_set_pi64x* | _m_paddsw | _mm_slli_pi32 | _mm_xor_si64 |
| _mm_cvtsi64_si32 | _mm_adds_pu8 | _m_pslldi | _m_pxor |
| _m_to_int | _m_paddusb | _mm_sll_si64 | _mm_cmpeq_pi8 |
| _mm_cvtsi64_si64x* | _mm_adds_pu16 | _m_psllq | _m_pcmpeqb |
| _mm_packs_pi16* | _m_paddusw | _mm_slli_si64 | _mm_cmpgt_pi8 |
| _m_packsswb | _mm_sub_pi8 | _m_psllqi | _m_pcmpgtb |
| _mm_packs_pi32 | _m_psubb | _mm_sra_pi16 | _mm_cmpeq_pi16 |
| _m_packssdw | _mm_sub_pi16 | _m_psraw | _m_pcmpeqw |
| _mm_packs_pu16 | _m_psubw | _mm_srai_pi16 | _mm_cmpgt_pi16 |
| _m_packuswb | _mm_sub_pi32 | _m_psrawi | _m_pcmpgtw |
| _mm_unpackhi_pi8 | _m_psubd | _mm_sra_pi32 | _mm_cmpeq_pi32 |
| _m_punpckhbw | _mm_sub_si64 | _m_psrad | _m_pcmpeqd |
| _mm_unpackhi_pi16 | _mm_subs_pi8 | _mm_srai_pi32 | _mm_cmpgt_pi32 |
| _m_punpckhwd | _m_psubsb | _m_psradi | _m_pcmpgtd |

continues on next page

Table 1 – continued from previous page

| | | | |
|---|---|---|---|
| _mm_unpackhi_pi32 | _mm_subs_pi16 | _mm_srl_pi16 | _mm_setzero_si64 |
| _m_punpckhdq | _m_psubsw | _m_psrlw | _mm_set_pi32 |
| _mm_unpacklo_pi8 | _mm_subs_pu8 | _mm_srli_pi16 | _mm_set_pi16 |
| _m_punpcklbw | _m_psubusb | _m_psrlwi | _mm_set_pi8 |
| _mm_unpacklo_pi16 | _mm_subs_pu16 | _mm_srl_pi32 | _mm_setr_pi32 |
| _m_punpcklwd | _m_psubusw | _m_psrld | _mm_setr_pi16 |
| _mm_unpacklo_pi32 | _mm_madd_pi16 | _mm_srli_pi32 | _mm_setr_pi8 |
| _m_punpckldq | _m_pmaddwd | _m_psrldi | _mm_set1_pi32 |
| _mm_add_pi8 | _mm_mulhi_pi16 | _mm_srl_si64 | _mm_set1_pi16 |
| _m_paddb | _m_pmulhw | _m_psrlq | _mm_set1_pi8 |
| _mm_add_pi16 | _mm_mullo_pi16 | _mm_srli_si64 | |
| _m_paddw | _m_pmullw | _m_psrlqi | |
| _mm_add_pi32 | _mm_sll_pi16 | _mm_and_si64 | |

## 8.3. x86-64 SSE Intrinsics

NVC++ and NVC support a set of SSE Intrinsics which allows the use of the SSE instructions directly from C++ and C code, without writing the assembly instructions. The following tables list the SSE intrinsics supported.

**Note:** Intrinsics with a * are only available on 64-bit systems.

Table 2: Table 22. SSE Intrinsics (xmmintrin.h)

| | | |
|---|---|---|
| _mm_add_ss | _mm_comige_ss | _mm_load_ss |
| _mm_sub_ss | _mm_comineq_ss | _mm_load1_ps |
| _mm_mul_ss | _mm_ucomieq_ss | _mm_load_ps1 |
| _mm_div_ss | _mm_ucomilt_ss | _mm_load_ps |
| _mm_sqrt_ss | _mm_ucomile_ss | _mm_loadu_ps |
| _mm_rcp_ss | _mm_ucomigt_ss | _mm_loadr_ps |
| _mm_rsqrt_ss | _mm_ucomige_ss | _mm_set_ss |
| _mm_min_ss | _mm_ucomineq_ss | _mm_set1_ps |
| _mm_max_ss | _mm_cvtss_si32 | _mm_set_ps1 |
| _mm_add_ps | _mm_cvt_ss2si | _mm_set_ps |

continues on next page

Table 2 – continued from previous page

| | | |
|---|---|---|
| _mm_sub_ps | _mm_cvtss_si64x* | _mm_setr_ps |
| _mm_mul_ps | _mm_cvtps_pi32 | _mm_store_ss |
| _mm_div_ps | _mm_cvt_ps2pi | _mm_store_ps |
| _mm_sqrt_ps | _mm_cvttss_si32 | _mm_store1_ps |
| _mm_rcp_ps | _mm_cvtt_ss2si | _mm_store_ps1 |
| _mm_rsqrt_ps | _mm_cvttss_si64x* | _mm_storeu_ps |
| _mm_min_ps | _mm_cvttps_pi32 | _mm_storer_ps |
| _mm_max_ps | _mm_cvtt_ps2pi | _mm_move_ss |
| _mm_and_ps | _mm_cvtsi32_ss | _mm_extract_pi16 |
| _mm_andnot_ps | _mm_cvt_si2ss | _m_pextrw |
| _mm_or_ps | _mm_cvtsi64x_ss* | _mm_insert_pi16 |
| _mm_xor_ps | _mm_cvtpi32_ps | _m_pinsrw |
| _mm_cmpeq_ss | _mm_cvt_pi2ps | _mm_max_pi16 |
| _mm_cmplt_ss | _mm_movelh_ps | _m_pmaxsw |
| _mm_cmple_ss | _mm_setzero_ps | _mm_max_pu8 |
| _mm_cmpgt_ss | _mm_cvtpi16_ps | _m_pmaxub |
| _mm_cmpge_ss | _mm_cvtpu16_ps | _mm_min_pi16 |
| _mm_cmpneq_ss | _mm_cvtpi8_ps | _m_pminsw |
| _mm_cmpnlt_ss | _mm_cvtpu8_ps | _mm_min_pu8 |
| _mm_cmpnle_ss | _mm_cvtpi32x2_ps | _m_pminub |
| _mm_cmpngt_ss | _mm_movehl_ps | _mm_movemask_pi8 |
| _mm_cmpnge_ss | _mm_cvtps_pi16 | _m_pmovmskb |
| _mm_cmpord_ss | _mm_cvtps_pi8 | _mm_mulhi_pu16 |
| _mm_cmpunord_ss | _mm_shuffle_ps | _m_pmulhuw |
| _mm_cmpeq_ps | _mm_unpackhi_ps | _mm_shuffle_pi16 |
| _mm_cmplt_ps | _mm_unpacklo_ps | _m_pshufw |
| _mm_cmple_ps | _mm_loadh_pi | _mm_maskmove_si64 |
| _mm_cmpgt_ps | _mm_storeh_pi | _m_maskmovq |
| _mm_cmpge_ps | _mm_loadl_pi | _mm_avg_pu8 |
| _mm_cmpneq_ps | _mm_storel_pi | _m_pavgb |
| _mm_cmpnlt_ps | _mm_movemask_ps | _mm_avg_pu16 |
| _mm_cmpnle_ps | _mm_getcsr | _m_pavgw |
| _mm_cmpngt_ps | _MM_GET_EXCEPTION_STATE | _mm_sad_pu8 |

continues on next page

Table 2 – continued from previous page

| | | |
|---|---|---|
| _mm_cmpnge_ps | _MM_GET_EXCEPTION_MASK | _m_psadbw |
| _mm_cmpord_ps | _MM_GET_ROUNDING_MODE | _mm_prefetch |
| _mm_cmpunord_ps | _MM_GET_FLUSH_ZERO_MODE | _mm_stream_pi |
| _mm_comieq_ss | _mm_setcsr | _mm_stream_ps |
| _mm_comilt_ss | _MM_SET_EXCEPTION_STATE | _mm_sfence |
| _mm_comile_ss | _MM_SET_EXCEPTION_MASK | _mm_pause |
| _mm_comigt_ss | _MM_SET_ROUNDING_MODE _MM_SET_FLUSH_ZERO_MODE | _MM_TRANSPOSE4_PS |

*Table 23* lists the SSE2 intrinsics that are supported and available in emmintrin.h.

Table 3: Table 23. SSE2 Intrinsics (emmintrin.h)

| | | | |
|---|---|---|---|
| _mm_load_sd | _mm_cmpge_sd | _mm_cvtps_pd | _mm_srl_epi32 |
| _mm_load1_pd | _mm_cmpneq_sd | _mm_cvtsd_si32 | _mm_srl_epi64 |
| _mm_load_pd1 | _mm_cmpnlt_sd | _mm_cvtsd_si64x* | _mm_slli_epi16 |
| _mm_load_pd | _mm_cmpnle_sd | _mm_cvttsd_si32 | _mm_slli_epi32 |
| _mm_loadu_pd | _mm_cmpngt_sd | _mm_cvttsd_si64x* | _mm_slli_epi64 |
| _mm_loadr_pd | _mm_cmpnge_sd | _mm_cvtsd_ss | _mm_srai_epi16 |
| _mm_set_sd | _mm_cmpord_sd | _mm_cvtsi32_sd | _mm_srai_epi32 |
| _mm_set1_pd | _mm_cmpunord_sd | _mm_cvtsi64x_sd* | _mm_srli_epi16 |
| _mm_set_pd1 | _mm_comieq_sd | _mm_cvtss_sd | _mm_srli_epi32 |
| _mm_set_pd | _mm_comilt_sd | _mm_unpackhi_pd | _mm_srli_epi64 |
| _mm_setr_pd | _mm_comile_sd | _mm_unpacklo_pd | _mm_and_si128 |
| _mm_setzero_pd | _mm_comigt_sd | _mm_loadh_pd | _mm_andnot_si128 |
| _mm_store_sd | _mm_comige_sd | _mm_storeh_pd | _mm_or_si128 |
| _mm_store_pd | _mm_comineq_sd | _mm_loadl_pd | _mm_xor_si128 |
| _mm_store1_pd | _mm_ucomieq_sd | _mm_storel_pd | _mm_cmpeq_epi8 |
| _mm_store_pd1 | _mm_ucomilt_sd | _mm_movemask_pd | _mm_cmpeq_epi16 |
| _mm_storeu_pd | _mm_ucomile_sd | _mm_packs_epi16 | _mm_cmpeq_epi32 |
| _mm_storer_pd | _mm_ucomigt_sd | _mm_packs_epi32 | _mm_cmplt_epi8 |
| _mm_move_sd | _mm_ucomige_sd | _mm_packus_epi16 | _mm_cmplt_epi16 |
| _mm_add_pd | _mm_ucomineq_sd | _mm_unpackhi_epi8 | _mm_cmplt_epi32 |
| _mm_add_sd | _mm_load_si128 | _mm_unpackhi_epi16 | _mm_cmpgt_epi8 |
| _mm_sub_pd | _mm_loadu_si128 | _mm_unpackhi_epi32 | _mm_cmpgt_epi16 |
| _mm_sub_sd | _mm_loadl_epi64 | _mm_unpackhi_epi64 | _mm_srl_epi16 |

continues on next page

Table 3 – continued from previous page

| | | | |
|---|---|---|---|
| _mm_mul_pd | _mm_store_si128 | _mm_unpacklo_epi8 | _mm_cmpgt_epi32 |
| _mm_mul_sd | _mm_storeu_si128 | _mm_unpacklo_epi16 | _mm_max_epi16 |
| _mm_div_pd | _mm_storel_epi64 | _mm_unpacklo_epi32 | _mm_max_epu8 |
| _mm_div_sd | _mm_movepi64_pi64 | _mm_unpacklo_epi64 | _mm_min_epi16 |
| _mm_sqrt_pd | _mm_move_epi64 | _mm_add_epi8 | _mm_min_epu8 |
| _mm_sqrt_sd | _mm_setzero_si128 | _mm_add_epi16 | _mm_movemask_epi8 |
| _mm_min_pd | _mm_set_epi64 | _mm_add_epi32 | _mm_mulhi_epu16 |
| _mm_min_sd | _mm_set_epi32 | _mm_add_epi64 | _mm_maskmoveu_si128 |
| _mm_max_pd | _mm_set_epi64x* | _mm_adds_epi8 | _mm_avg_epu8 |
| _mm_max_sd | _mm_set_epi16 | _mm_adds_epi16 | _mm_avg_epu16 |
| _mm_and_pd | _mm_set_epi8 | _mm_adds_epu8 | _mm_sad_epu8 |
| _mm_andnot_pd | _mm_set1_epi64 | _mm_adds_epu16 | _mm_stream_si32 |
| _mm_or_pd | _mm_set1_epi32 | _mm_sub_epi8 | _mm_stream_si128 |
| _mm_xor_pd | _mm_set1_epi64x* | _mm_sub_epi16 | _mm_stream_pd |
| _mm_cmpeq_pd | _mm_set1_epi16 | _mm_sub_epi32 | _mm_movpi64_epi64 |
| _mm_cmplt_pd | _mm_set1_epi8 | _mm_sub_epi64 | _mm_lfence |
| _mm_cmple_pd | _mm_setr_epi64 | _mm_subs_epi8 | _mm_mfence |
| _mm_cmpgt_pd | _mm_setr_epi32 | _mm_subs_epi16 | _mm_cvtsi32_si128 |
| _mm_cmpge_pd | _mm_setr_epi16 | _mm_subs_epu8 | _mm_cvtsi64x_si128* |
| _mm_cmpneq_pd | _mm_setr_epi8 | _mm_subs_epu16 | _mm_cvtsi128_si32 |
| _mm_cmpnlt_pd | _mm_cvtepi32_pd | _mm_madd_epi16 | _mm_cvtsi128_si64x* |
| _mm_cmpnle_pd | _mm_cvtepi32_ps | _mm_mulhi_epi16 | _mm_srli_si128 |
| _mm_cmpngt_pd | _mm_cvtpd_epi32 | _mm_mullo_epi16 | _mm_slli_si128 |
| _mm_cmpnge_pd | _mm_cvtpd_pi32 | _mm_mul_su32 | _mm_shuffle_pd |
| _mm_cmpord_pd | _mm_cvtpd_ps | _mm_mul_epu32 | _mm_shufflehi_epi16 |
| _mm_cmpunord_pd | _mm_cvttpd_epi32 | _mm_sll_epi16 | _mm_shufflelo_epi16 |
| _mm_cmpeq_sd | _mm_cvttpd_pi32 | _mm_sll_epi32 | _mm_shuffle_epi32 |
| _mm_cmplt_sd | _mm_cvtpi32_pd | _mm_sll_epi64 | _mm_extract_epi16 |
| _mm_cmple_sd | _mm_cvtps_epi32 | _mm_sra_epi16 | _mm_insert_epi16 |
| _mm_cmpgt_sd | _mm_cvttps_epi32 | _mm_sra_epi32 | |

*Table 24* lists the SSE3 intrinsics supported and available in pmmintrin.h.

Table 4: Table 24. SSE3 Intrinsics (pmmintrin.h)

| | | | |
|---|---|---|---|
| _mm_addsub_ps | _mm_moveldup_ps | _mm_loaddup_pd | _mm_mwait |
| _mm_hadd_ps | _mm_addsub_pd | _mm_movedup_pd | |
| _mm_hsub_ps | _mm_hadd_pd | _mm_lddqu_si128 | |
| _mm_movehdup_ps | _mm_hsub_pd | _mm_monitor | |

*Table 25* lists the SSSE3 intrinsics supported and available in tmmintrin.h.

Table 5: Table 25. SSSE3 Intrinsics (tmmintrin.h)

| | | |
|---|---|---|
| _mm_hadd_epi16 | _mm_hsubs_pi16 | _mm_sign_pi16 |
| _mm_hadd_epi32 | _mm_maddubs_epi16 | _mm_sign_pi32 |
| _mm_hadds_epi16 | _mm_maddubs_pi16 | _mm_alignr_epi8 |
| _mm_hadd_pi16 | _mm_mulhrs_epi16 | _mm_alignr_pi8 |
| _mm_hadd_pi32 | _mm_mulhrs_pi16 | _mm_abs_epi8 |
| _mm_hadds_pi16 | _mm_shuffle_epi8 | _mm_abs_epi16 |
| _mm_hsub_epi16 | _mm_shuffle_pi8 | _mm_abs_epi32 |
| _mm_hsub_epi32 | _mm_sign_epi8 | _mm_abs_pi8 |
| _mm_hsubs_epi16 | _mm_sign_epi16 | _mm_abs_pi16 |
| _mm_hsub_pi16 | _mm_sign_epi32 | _mm_abs_pi32 |
| _mm_hsub_pi32 | _mm_sign_pi8 | |

*Table 26* lists the SSE4a intrinsics supported and available in ammintrin.h.

Table 6: Table 26. SSE4a Intrinsics (ammintrin.h)

| | | |
|---|---|---|
| _mm_stream_sd | _mm_extract_si64 | _mm_insert_si64 |
| _mm_stream_ss | _mm_extracti_si64 | _mm_inserti_si64 |

## 8.4. x86-64 ABM Intrinsics

NVC++ and NVC support a set of ABM Intrinsics which allow the use of the ABM instructions directly from C++ and C code, without writing the assembly instructions. The following table lists the ABM intrinsics supported.

Table 7: Table 27. ABM Intrinsics (intrin.h)

| | | | |
|---|---|---|---|
| __lzcnt16 | __lzcnt64 | __popcnt | __rdtscp |
| __lzcnt | __popcnt16 | __popcnt64 | |

# 8.5. x86-64 AVX Intrinsics

The following table lists the AVX intrinsics supported by NVC++ and NVC.

Table 8: Table 28. AVX Intrinsics (immintrin.h)

| | | |
|---|---|---|
| _mm256_add_pd | _mm256_add_ps | _mm256_addsub_pd |
| _mm256_addsub_ps | _mm256_and_pd | _mm256_and_ps |
| _mm256_andnot_pd | _mm256_andnot_ps | _mm256_blendv_pd |
| _mm256_blendv_ps | _mm256_broadcast_pd | _mm256_broadcast_ps |
| _mm256_broadcast_sd | _mm256_broadcast_ss | _mm256_castpd_si256 |
| _mm256_castps_si256 | _mm256_castpd_ps | _mm256_castps_pd |
| _mm256_castpd128_pd256 | _mm256_castpd256_pd128 | _mm256_castsi256_pd |
| _mm256_castsi256_ps | _mm256_cvtepi32_pd | _mm256_cvtepi32_ps |
| _mm256_cvtpd_epi32 | _mm256_cvtps_epi32 | _mm256_cvtpd_ps |
| _mm256_cvtps_pd | _mm256_cvttpd_epi32 | _mm256_cvttps_epi32 |
| _mm256_div_pd | _mm256_div_ps | _mm256_hadd_pd |
| _mm256_hadd_ps | _mm256_hsub_pd | _mm256_hsub_ps |
| _mm256_load_pd | _mm256_load_ps | _mm256_loadu_pd |
| _mm256_loadu_ps | _mm256_maskload_pd | _mm256_maskload_ps |
| _mm256_maskstore_pd | _mm256_maskstore_ps | _mm256_max_pd |
| _mm256_max_ps | _mm256_min_pd | _mm256_min_ps |
| _mm256_movemask_pd | _mm256_movemask_ps | _mm256_mul_pd |
| _mm256_mul_ps | _mm256_or_pd | _mm256_or_ps |
| _mm256_rcp_ps | _mm256_rsqrt_ps | _mm256_set_pd |
| _mm256_set_ps | _mm256_setr_pd | _mm256_setr_ps |
| _mm256_set1_pd | _mm256_set1_ps | _mm256_set_epi32 |
| _mm256_set_epi64x | _mm256_setzero_pd | _mm256_setzero_ps |
| _mm256_sqrt_pd | _mm256_sqrt_ps | _mm256_store_pd |
| _mm256_store_ps | _mm256_storeu_pd | _mm256_storeu_ps |
| _mm256_stream_pd | _mm256_stream_ps | _mm256_stream_si256 |
| _mm256_sub_pd | _mm256_sub_ps | _mm256_testz_pd |
| _mm256_testz_ps | _mm256_testc_pd | _mm256_testc_ps |
| _mm256_testnzc_pd | _mm256_testnzc_ps | _mm256_unpackhi_pd |
| _mm256_unpackhi_ps | _mm256_unpacklo_pd | _mm256_unpacklo_ps |

continues on next page

Table 8 – continued from previous page

| | | |
|---|---|---|
| _mm256_xor_pd | _mm256_xor_ps | _mm256_zeroupper |
| _mm256_macc_pd | _mm256_macc_ps | _mm256_msub_pd |
| _mm256_msub_ps | _mm256_nmacc_pd | _mm256_nmacc_ps |
| _mm256_nmsub_pd | _mm256_nmsub_ps | _mm256_maddsub_pd |
| _mm256_maddsub_ps | _mm256_msubadd_pd | _mm256_msubadd_ps |
| _mm_macc_pd | _mm_macc_ps | _mm_msub_pd |
| _mm_msub_ps | _mm_nmacc_pd | _mm_nmacc_ps |
| _mm_nmsub_pd | _mm_nmsub_ps | _mm_maddsub_pd |
| _mm_maddsub_ps | _mm_msubadd_pd | _mm_msubadd_ps |
| _mm_macc_sd | _mm_macc_ss | _mm_msub_sd |
| _mm_msub_ss | _mm_nmacc_sd | _mm_nmacc_ss |
| _mm_nmsub_sd | _mm_nmsub_ss | _mm256_extractf128_pd |
| _mm256_extractf128_ps | _mm256_extractf128_si256 | _mm256_permute_pd |
| _mm256_permute_ps | _mm256_permute2f128_pd | _mm256_permute2f128_ps |
| _mm256_permute2f128_si256 | _mm256_blend_pd | _mm256_blend_ps |
| _mm256_shuffle_pd | _mm256_shuffle_ps | _mm256_cmp_pd |
| _mm256_cmp_ps | _mm256_round_pd | _mm256_round_ps |
| _mm256_insertf128_pd | _mm256_insertf128_ps | _mm256_insertf128_si256 |
| _mm256_dp_ps | | |

# Chapter 9. Messages

This section describes the various messages that the compiler produces. These messages include the sign-on message and diagnostic messages for remarks, warnings, and errors. The compiler always displays any error messages, along with the erroneous source line, on the screen. If you specify the `-Mlist` option, the compiler places any error messages in the listing file. You can also use the `-v` option to display more information about the compiler, assembler, and linker invocations and about the host system. For more information on the `-Mlist` and `-v` options, refer to 'Using Command-line Options' in the HPC Compiler User Guide.

## 9.1. Diagnostic Messages

Diagnostic messages provide syntactic and semantic information about your source text. Syntactic information includes information such as syntax errors. Semantic information includes information such as unreachable code, incorrect number of arguments specified for a call to a routine, illegal data type usage, etc.

You can specify that the compiler displays error messages at a certain level with the `-Minform` option.

The compiler messages refer to a severity level, a message number, and the line number where the error occurs.

The compiler can also display internal error messages on standard error.

If you use the listing file option `-Mlist`, the compiler places diagnostic messages after the source lines in the listing file, in the following format:

```
NVFORTRAN-etype-enum-message (filename: line)
```

Where:

**etype**
: is a character signifying the severity level

**enum**
: is the error number

**message**
: is the error message

**filename**
: is the source filename

**line** is the line number where the compiler detected an error.

## 9.2. Phase Invocation Messages

You can display compiler, assembler, and linker phase invocations by using the -v command line option. For further information about this option, refer to the 'Using Command-line Options' section of the HPC Compilers User Guide.

## 9.3. Fortran Compiler Error Messages

This section presents the error messages generated by the *NVFORTRAN* compiler. The compiler displays error messages in the program listing and on standard output. They can also display internal compiler error messages on standard error.

### 9.3.1. Message Format

Each message is numbered. Each message also lists the line and column number where the error occurs. A dollar sign ($) in a message represents information that is specific to each occurrence of the message.

### 9.3.2. Message List

Error message severities:

**I** informative

**W** warning

**S** severe error

**F** fatal error

**V** variable

```
V000 Internal compiler error. $ $
```

This message indicates an error in the compiler, rather than a user error – although it may be possible for a user error to cause an internal error. The severity may vary; if it is informative or warning, correct object code was probably generated, but it is not safe to rely on this.

```
F001 Source input file name not specified
```

On the command line, source file name should be specified either before all the switches, or after them.

```
F002 Unable to open source input file: $
```

Source file name is misspelled, file is not in current working directory, or file is read protected.

```
F003 Unable to open listing file
```

This message typically occurs when the user does not have write permission for the current working directory.

```
F004 $ $
```

Generic message for file errors.

```
F005 Unable to open temporary file
```

Compiler uses directory “/usr/tmp” or “/tmp” in which to create temporary files. If neither of these directories is available on the node on which the compiler is being used, this error will occur.

```
S006 Input file empty
```

Source input file does not contain any Fortran statements other than comments or compiler directives.

```
F007 Subprogram too large to compile at this optimization level $
```

Internal compiler data structure overflow, working storage exhausted, or some other non-recoverable problem related to the size of the subprogram. If this error occurs at opt level 2, reducing the opt level to 1 may work around the problem. Moving the subprogram being compiled to its own source file may eliminate the problem.

```
F008 Error limit exceeded
```

The compiler gives up because too many severe errors were issued; the error limit can be reset on the command line.

```
F009 Unable to open assembly file
```

This message typically occurs when the user does not have write permission for the current working directory.

```
F010 File write error occurred $
```

The file system may be full.

```
S011 Unrecognized command line switch: $
```

Refer to the HPC Compiler User Guide for a list of allowed compiler switches.

```
S012 Value required for command line switch: $
```

Certain switches require an immediately following value, such as “-opt 2”.

```
S013 Unrecognized value specified for command line switch: $
```

```
S014 Ambiguous command line switch: $
```

Too short an abbreviation was used for one of the switches.

```
W015 Hexadecimal or octal constant truncated to fit data type
```

```
I016 Identifier, $, truncated to 63 chars
```

An identifier may be at most 63 characters in length; characters after the 63rd are ignored.

```
S017 Unable to open include file: $
```

File is missing, read protected, or maximum include depth (10) exceeded. Remember that the file name should be enclosed in quotes.

```
S018 Illegal label $ $
```

Used for label 'field' errors or illegal values. E.g., in fixed source form, the label field (first five characters) of the indicated line contains a non-numeric character.

```
S019 Illegally placed continuation line
```

A continuation line does not follow an initial line, or more than 99 continuation lines were specified.

```
S020 Unrecognized compiler directive
```

```
S021 Label field of continuation line is not blank
```

The first five characters of a continuation line must be blank.

```
S022 Unexpected end of file - missing END statement
```

The source file is missing and END statement, or the file is truncated.

```
S023 Syntax error - unbalanced $
```

Unbalanced parentheses or brackets.

```
W024 CHARACTER or Hollerith constant truncated to fit data type
```

A character or hollerith constant was converted to a data type that was not large enough to contain all of the characters in the constant. This type conversion occurs when the constant is used in an arithmetic expression or is assigned to a non-character variable. The character or hollerith constant is truncated on the right, that is, if 4 characters are needed then the first 4 are used and the remaining characters are discarded.

```
W025 Illegal character ($) - ignored
```

The current line contains a character, possibly non-printing, which is not a legal Fortran character (characters inside of character or Hollerith constants cannot cause this error). As a general rule, all non-printing characters are treated as white space characters (blanks and tabs); no error message is generated when this occurs. If for some reason, a non-printing character is not treated as a white space character, its hex representation is printed in the form dd where each d is a hex digit.

```
S026 Unmatched quote
```

A character constant is missing a closing quote or the source file is truncated.

```
S027 Illegal integer constant: $
```

Integer constant is too large for 32 bit word.

```
S028 Illegal real or double precision constant: $
```

```
S029 Illegal $ constant: $
```

Illegal hexadecimal, octal, or binary constant. A hexadecimal constant consists of digits 0..9 and letters A..F or a..f; any other character in a hexadecimal constant is illegal. An octal constant consists of digits 0..7; any other digit or character in an octal constant is illegal. A binary constant consists of digits 0 or 1; any other digit or character in a binary constant is illegal.

```
S030 Explicit shape must be specified for $
```

A shape for an array expression is effected in this context.

```
S031 Illegal data type length specifier for $
```

The data type length specifier (e.g. 4 in INTEGER*4) is not a constant expression that is a member of the set of allowed values for this particular data type.

```
W032 Data type length specifier not allowed for $
```

The data type length specifier (e.g. 4 in INTEGER*4) is not allowed in the given syntax (e.g. DIMENSION A(10)*4).

```
S033 Illegal use of constant $
```

A constant was used in an illegal context, such as on the left side of an assignment statement or as the target of a data initialization statement.

```
S034 Syntax error at or near $
```

Illegal command specified.

```
I035 Predefined intrinsic $ loses intrinsic property
```

An intrinsic name was used in a manner inconsistent with the language definition for that intrinsic. The compiler, based on the context, will treat the name as a variable or an external function.

```
S036 Illegal implicit character range
```

First character must alphabetically precede second.

```
S037 Contradictory data type specified for $
```

The indicated identifier appears in more than one type specification statement and different data types are specified for it.

```
S038 Symbol, $, has not been explicitly declared
```

The indicated identifier must be declared in a type statement; this is required when the IMPLICIT NONE statement occurs in the subprogram.

```
W039 Symbol, $, appears illegally in a SAVE statement $
```

An identifier appearing in a SAVE statement must be a local variable or array.

```
S040 Illegal common variable $
```

Indicated identifier is a dummy variable, is already in a common block, or has previously been defined to be something other than a variable or array.

```
W041 Illegal use of dummy argument $
```

This error can occur in several situations. It can occur if dummy arguments were specified on a PROGRAM statement. It can also occur if a dummy argument name occurs in a DATA, COMMON, SAVE, or EQUIVALENCE statement. A program statement must have an empty argument list.

```
S042 $ is a duplicate dummy argument
```

Each dummy argument must have a unique name.

```
S043 Illegal attempt to redefine $ $
```

An attempt was made to define a symbol in a manner inconsistent with an earlier definition of the same symbol. This can happen for a number of reasons. The message attempts to indicate the situation that occurred.

intrinsic – An attempt was made to redefine an intrinsic function. A symbol that represents an intrinsic function may be redefined if that symbol has not been previously verified to be an intrinsic function. For example, the intrinsic `sin` can be defined to be an integer array. If a symbol is verified to be an intrinsic function via the INTRINSIC statement or via an intrinsic function reference then it must be referred to as an intrinsic function for the remainder of the program unit.

symbol – An attempt was made to redefine a symbol that was previously defined. An example of this is to declare a symbol to be a PARAMETER which was previously declared to be a subprogram argument.

```
S044 Multiple declaration for symbol $
```

A redundant declaration of a symbol has occurred. For example, an attempt was made to declare a symbol as an ENTRY when that symbol was previously declared as an ENTRY.

```
S045 Data type of entry point $ disagrees with function $
```

The current function has entry points with data types inconsistent with the data type of the current function. For example, the function returns type character and an entry point returns type complex.

```
S046 Data type length specifier in wrong position
```

The CHARACTER data type specifier has a different position for the length specifier from the other data types. Suppose, we want to declare arrays ARRAYA and ARRAYB to have 8 elements each having an element length of 4 bytes. The difference is that ARRAYA is character and ARRAYB is integer. The declarations would be CHARACTER ARRAYA(8)*4 and INTEGER ARRAYB*4(8).

```
S047 More than seven dimensions specified for array
```

The compiler currently supports up to seven dimensions for arrays.

```
S048 Illegal use of '*' in declaration of array $
```

An asterisk may be used only as the upper bound of the last dimension.

```
S049 Illegal use of '*' in non-subroutine subprogram
```

The alternate return specifier '*' is legal only in the subroutine statement. Programs, functions, and block data are not allowed to have alternate return specifiers.

```
S050 Assumed size array, $, is not a dummy argument
```

Arrays with '*' in their dimension(s) may only be declared as dummy arguments.

```
S051 Unrecognized built-in % function
```

The allowable built-in functions are %VAL, %REF, %LOC, and %FILL. One was encountered that did not match one of these allowed forms.

```
S052 Illegal argument to %VAL or %LOC
```

```
S053 %REF or %VAL not legal in this context
```

The built-in functions %REF and %VAL can only be used as actual parameters in procedure calls.

```
W054 Implicit character $ used in a previous implicit statement
```

An implicit character has been given an implied data type more than once. The implied data type for the implicit character is changed anyway.

```
W055 Multiple implicit none statements
```

The IMPLICIT NONE statement can occur only once in a subprogram.

```
W056 Implicit type declaration
```

The -Mdclchk switch and an implicit declaration following an IMPLICIT NONE statement will produce a warning message for IMPLICIT statements.

```
S057 Illegal equivalence of dummy variable, $
```

Dummy arguments may not appear in EQUIVALENCE statements.

```
S058 Equivalenced variables $ and $ not in same common block
```

A common block variable must not be equivalenced with a variable in another common block.

```
S059 Conflicting equivalence between $ and $
```

The indicated equivalence implies a storage layout inconsistent with other equivalences.

```
S060 Illegal equivalence of structure variable, $
```

STRUCTURE and UNION variables may not appear in EQUIVALENCE statements.

```
S061 Equivalence of $ and $ extends common block backwards
```

```
W062 Equivalence forces $ to be unaligned
```

EQUIVALENCE statements have defined an address for the variable which has an alignment not optimal for variables of its data type. This can occur when INTEGER and CHARACTER data are equivalenced, for instance.

```
I063 Gap in common block $ before $
```

```
S064 Illegal use of $ in DATA statement implied DO loop
```

The indicated variable is referenced where it is not an active implied DO index variable.

```
S065 Repeat factor less than zero
```

```
S066 Too few data constants in initialization statement
```

```
S067 Too many data constants in initialization statement
```

```
S068 Numeric initializer for CHARACTER $ out of range 0 through 255
```

A CHARACTER*1 variable or character array element can be initialized to an integer, octal, or hexadecimal constant if that constant is in the range 0 through 255.

```
S069 Illegal implied DO expression
```

The only operations allowed within an implied DO expression are integer +, -, *, and /.

```
S070 Incorrect sequence of statements $
```

The statement order is incorrect. For instance, an IMPLICIT NONE statement must precede a specification statement which in turn must precede an executable statement.

```
S071 Executable statements not allowed in block data
```

```
S072 Assignment operation illegal to $ $
```

The destination of an assignment operation must be a variable, array reference, or vector reference. The assignment operation may be by way of an assignment statement, a data statement, or the index variable of an implied DO-loop. The compiler has determined that the identifier used as the destination is not a storage location. The error message attempts to indicate the type of entity used.

**entry point** – An assignment to an entry point that was not a function procedure was attempted.

**external procedure** – An assignment to an external procedure or a Fortran intrinsic name was attempted. If the identifier is the name of an entry point that is not a function, an external procedure.

```
S073 Intrinsic or predeclared, $, cannot be passed as an argument
```

```
S074 Illegal number or type of arguments to $ $
```

The indicated symbol is an intrinsic or generic function, or a predeclared subroutine or function, requiring a certain number of arguments of a fixed data type.

```
S075 Subscript, substring, or argument illegal in this context for $
```

This can happen if you try to doubly index an array such as ra(2)(3). This also applies to substring and function references.

```
S076 Subscripts specified for non-array variable $
```

```
S077 Subscripts omitted from array $
```

```
S078 Wrong number of subscripts specified for $
```

```
S079 Keyword form of argument illegal in this context for $
```

```
S080 Subscript for array $ is out of bounds
```

```
S081 Illegal selector $ $
```

```
S082 Illegal substring expression for variable $
```

Substring expressions must be of type integer and if constant must be greater than zero.

```
S083 Vector expression used where scalar expression required
```

A vector expression was used in an illegal context. For example, `iscalar = iarray`, where a scalar is assigned the value of an array. Also, character and record references are not vectorizable.

```
S084 Illegal use of symbol $ $
```

This message is used for many different errors.

```
S085 Incorrect number of arguments to statement function $
```

```
S086 Dummy argument to statement function must be a variable
```

```
S087 Non-constant expression where constant expression required
```

```
S088 Recursive subroutine or function call of $
```

A function may not call itself.

```
S089 Illegal use of symbol, $, with character length = *
```

Symbols of type CHARACTER*(*) must be dummy variables and must not be used as statement function dummy parameters and statement function names. Also, a dummy variable of type CHARACTER*(*) cannot be used as a function.

```
S090 Hollerith constant more than 4 characters
```

In certain contexts, Hollerith constants may not be more than 4 characters long.

```
S091 Constant expression of wrong data type
```

```
S092 Illegal use of variable length character expression
```

A character expression used as an actual argument, or in certain contexts within I/O statements, must not consist of a concatenation involving a passed length character variable.

```
W093 Type conversion of expression performed
```

An expression of some data type appears in a context which requires an expression of some other data type. The compiler generates code to convert the expression into the required type.

```
S094 Variable $ is of wrong data type $
```

The indicated variable is used in a context which requires a variable of some other data type.

```
S095 Expression has wrong data type
```

An expression of some data type appears in a context which requires an expression of some other data type.

```
S096 Illegal complex comparison
```

The relations .LT., .GT., .GE., and .LE. are not allowed for complex values.

```
S097 Statement label $ has been defined more than once
```

More than one statement with the indicated statement number occurs in the subprogram.

```
S098 Divide by zero
```

```
S099 Illegal use of $
```

Aggregate record references may only appear in aggregate assignment statements, unformatted I/O statements, and as parameters to subprograms. They may not appear, for example, in expressions. Also, records with differing structure types may not be assigned to one another.

```
S100 Expression cannot be promoted to a vector
```

An expression was used that required a scalar quantity to be promoted to a vector illegally. For example, the assignment of a character constant string to a character array. Records, too, cannot be promoted to vectors.

```
S101 Vector operation not allowed on $
```

Record and character typed entities may only be referenced as scalar quantities.

```
S102 Arithmetic IF expression has wrong data type
```

The parenthetical expression of an arithmetic if statement must be an integer, real, or double precision scalar expression.

```
S103 Type conversion of subscript expression for $
```

The data type of a subscript expression must be integer. If it is not, it is converted.

```
S104 Illegal control structure $
```

This message is issued for a number of errors involving IF-THEN statements, DO loops, and directives. You may see one of the following messages:

```
NVFORTRAN-S-0104-Illegal control structure - unterminated PARALLEL directive
```

```
NVFORTRAN-S-0104-Illegal control structure - unterminated block IF
```

If the line number specified is the last line (END statement) of the subprogram, the error is probably an unterminated DO loop or IF-THEN statement. If the message contains `unterminated PARALLEL directive`, it is likely you are missing the required `!$omp end parallel` directive.

```
S105 Unmatched ELSEIF, ELSE or ENDIF statement
```

An ELSEIF, ELSE, or ENDIF statement cannot be matched with a preceding IF-THEN statement.

```
S106 DO index variable must be a scalar variable
```

The DO index variable cannot be an array name, a subscripted variable, a PARAMETER name, a function name, a structure name, etc.

```
S107 Illegal assigned goto variable $
```

```
S108 Illegal variable, $, in NAMELIST group $
```

A NAMELIST group can only consist of arrays and scalars.

```
I109 Overflow in $ constant $, constant truncated at left
```

A non-decimal (hexadecimal, octal, or binary) constant requiring more than 64-bits produces an overflow. The constant is truncated at left (e.g. '1234567890abcdef1'x will be '234567890abcdef1'x).

```
I110 <reserved message number>
```

```
I111 Underflow of real or double precision constant
```

```
I112 Overflow of real or double precision constant
```

```
S113 Label $ is referenced but never defined
```

```
S114 Cannot initialize $
```

```
W115 Assignment to DO variable $ in loop
```

```
S116 Illegal use of pointer-based variable $ $
```

```
S117 Statement not allowed within a $ definition
```

The statement may not appear in a STRUCTURE or derived type definition.

```
S118 Statement not allowed in DO, IF, or WHERE block
```

```
I119 Redundant specification for $
```

Data type of indicated symbol specified more than once.

```
I120 Label $ is defined but never referenced
```

```
I121 Operation requires logical or integer data types
```

An operation in an expression was attempted on data having a data type incompatible with the operation. For example, a logical expression can consist of only logical elements of type integer or logical. Real data would be invalid.

```
I122 Character string truncated
```

Character string or Hollerith constant appearing in a DATA statement or PARAMETER statement has been truncated to fit the declared size of the corresponding identifier.

```
W123 Hollerith length specification too big, reduced
```

The length specifier field of a hollerith constant specified more characters than were present in the character field of the hollerith constant. The length specifier was reduced to agree with the number of characters present.

```
S124 Relational expression mixes character with numeric data
```

A relational expression is used to compare two arithmetic expressions or two character expressions. A character expression cannot be compared to an arithmetic expression.

```
I125 Dummy procedure $ not declared EXTERNAL
```

A dummy argument which is not declared in an EXTERNAL statement is used as the subprogram name in a CALL statement, or is called as a function, and is therefore assumed to be a dummy procedure. This message can result from a failure to declare a dummy array.

```
I126 Name $ is not an intrinsic function
```

```
I127 Optimization level for $ changed to opt 1 $
```

```
W128 Integer constant truncated to fit data type: $
```

An integer constant will be truncated when assigned to data types smaller than 32-bits, such as a BYTE.

```
I129 Floating point overflow. Check constants and constant expressions
```

```
I130 Floating point underflow. Check constants and constant expressions
```

```
I131 Integer overflow. Check floating point expressions cast to integer
```

```
I132 Floating pt. invalid oprnd. Check constants and constant expressions
```

```
I133 Divide by 0.0. Check constants and constant expressions
```

```
S134 Illegal attribute $ $
```

```
W135 Missing STRUCTURE name field
```

A STRUCTURE name field is required on the outermost structure.

```
W136 Field-namelist not allowed
```

The field-namelist field of the STRUCTURE statement is disallowed on the outermost structure.

```
W137 Field-namelist is required in nested structures
```

```
W138 Multiply defined STRUCTURE member name $
```

A member name was used more than once within a structure.

```
W139 Structure $ in RECORD statement not defined
```

A RECORD statement contains a reference to a STRUCTURE that has not yet been defined.

```
S140 Variable $ is not a RECORD
```

```
S141 RECORD required on left of $
```

```
S142 $ is not a member of this RECORD
```

```
S143 $ requires initializer
```

```
W144 NEED ERROR MESSAGE $ $
```

This is used as a temporary message for compiler development.

```
W145 %FILL only valid within STRUCTURE block
```

The %FILL special name was used outside of a STRUCTURE multiline statement. It is only valid when used within a STRUCTURE multiline statement even though it is ignored.

```
S146 Expression must be character type
```

```
S147 Character expression not allowed in this context
```

```
S148 Reference to $ required
```

An aggregate reference to a record was expected during statement compilation but another data type was found instead.

```
S149 Record where arithmetic value required
```

An aggregate record reference was encountered when an arithmetic expression was expected.

```
S150 Structure, Record, derived type, or member $ not allowed in this context
```

A structure, record, or member reference was found in a context which is not supported.

```
S151 Empty TYPE, STRUCTURE, UNION, or MAP
```

TYPE - ENDTYPE, STRUCTURE - ENDSTRUCTURE, UNION - ENDUNION or MAP - ENDMAP declaration contains no members.

```
S152 All dimension specifiers must be ':'
```

```
S153 Array objects are not conformable $
```

```
S154 DISTRIBUTE target, $, must be a processor
```

```
S155 $ $
```

```
S156 Number of colons and triplets must be equal in ALIGN $ with $
```

```
S157 Illegal subscript use of ALIGN dummy $ - $
```

```
S158 Alternate return not specified in SUBROUTINE or ENTRY
```

An alternate return can only be used if alternate return specifiers appeared in the SUBROUTINE or ENTRY statements.

```
S159 Alternate return illegal in FUNCTION subprogram
```

An alternate return cannot be used in a FUNCTION.

```
S160 ENDSTRUCTURE, ENDUNION, or ENDMAP does not match top
```

```
S161 Vector subscript must be rank-one array
```

```
W162 Not equal test of loop control variable $ replaced with < or > test.
```

```
S163 <reserved message number>
```

```
S164 Overlapping data initializations of $
```

An attempt was made to data initialize a variable or array element already initialized.

```
S165 $ appeared more than once as a subprogram
```

A subprogram name appeared more than once in the source file. The message is applicable only when an assembly file is the output of the compiler.

```
S166 $ cannot be a common block and a subprogram
```

A name appeared as a common block name and a subprogram name. The message is applicable only when an assembly file is the output of the compiler.

```
I167 Inconsistent size of common block $
```

A common block occurs in more than one subprogram of a source file and its size is not identical. The maximum size is chosen. The message is applicable only when an assembly file is the output of the compiler.

```
S168 Incompatible size of common block $
```

A common block occurs in more than one subprogram of a source file and is initialized in one subprogram. Its initialized size was found to be less than its size in the other subprogram(s). The message is applicable only when an assembly file is the output of the compiler.

```
W169 Multiple data initializations of common block $
```

A common block is initialized in more than one subprogram of a source file. Only the first set of initializations apply. The message is applicable only when an assembly file is the output of the compiler.

```
W170 NVIDIA Fortran extension: $ $
```

Use of a nonstandard feature. A description of the feature is provided.

```
W171 NVIDIA Fortran extension: nonstandard statement type $
```

```
W172 NVIDIA Fortran extension: numeric initialization of CHARACTER $
```

A CHARACTER*1 variable or array element was initialized with a numeric value.

```
W173 NVIDIA Fortran extension: nonstandard use of data type length specifier
```

```
W174 NVIDIA Fortran extension: type declaration contains data initialization
```

```
W175 NVIDIA Fortran extension: IMPLICIT range contains nonalpha characters
```

```
W176 NVIDIA Fortran extension: nonstandard operator $
```

```
W177 NVIDIA Fortran extension: nonstandard use of keyword argument $
```

```
W178 <reserved message number>
```

```
W179 NVIDIA Fortran extension: use of structure field reference $
```

```
W180 NVIDIA Fortran extension: nonstandard form of constant
```

```
W181 NVIDIA Fortran extension: & alternate return
```

```
W182 NVIDIA Fortran extension: mixed non-character and character elements in COMMON $
```

```
W183 NVIDIA Fortran extension: mixed non-character and character EQUIVALENCE ($,$)
```

```
W184 Mixed type elements (numeric and/or character types) in COMMON $
```

```
W185 Mixed numeric and/or character type EQUIVALENCE ($,$)
```

```
S186 Argument missing for formal argument $
```

```
S187 Too many arguments specified for $
```

```
S188 Argument number $ to $: type mismatch
```

```
S189 Argument number $ to $: association of scalar actual argument to array dummy
↪argument
```

```
S190 Argument number $ to $: non-conformable arrays
```

```
S191 Argument number $ to $ cannot be an assumed-size array
```

```
S192 Argument number $ to $ must be a label
```

```
W193 Argument number $ to $ does not match INTENT (OUT)
```

```
W194 INTENT(IN) argument cannot be defined - $
```

```
S195 Statement may not appear in an INTERFACE block $
```

```
S196 Deferred-shape specifiers are required for $
```

```
S197 Invalid qualifier or qualifier value (/$) in OPTIONS statement
```

An illegal qualifier was found or a value was specified for a qualifier which does not expect a value. In either case, the qualifier for which the error occurred is indicated in the error message.

```
S198 $ $ in ALLOCATE/DEALLOCATE
```

```
W199 Unaligned memory reference
```

A memory reference occurred whose address does not meet its data alignment requirement.

```
S200 Missing UNIT/FILE specifier
```

```
S201 Illegal I/O specifier - $
```

```
S202 Repeated I/O specifier - $
```

```
S203 FORMAT statement has no label
```

```
S204 $ $
```

Miscellaneous I/O error.

```
S205 Illegal specification of scale factor
```

The integer following + or - has been omitted, or P does not follow the integer value.

```
S206 Repeat count is zero
```

```
S207 Integer constant expected in edit descriptor
```

```
S208 Period expected in edit descriptor
```

```
S209 Illegal edit descriptor
```

```
S210 Exponent width not used in the Ew.dEe or Gw.dEe edit descriptors
```

```
S211 Internal I/O not allowed in this I/O statement
```

```
S212 Illegal NAMELIST I/O
```

Namelist I/O cannot be performed with internal, unformatted, formatted, and list-directed I/O. Also, I/O lists must not be present.

```
S213 $ is not a NAMELIST group name
```

```
S214 Input item is not a variable reference
```

```
S215 Assumed sized array name cannot be used as an I/O item or specifier
```

An assumed size array was used as an item to be read or written or as an I/O specifier (i.e., FMT = array-name). In these contexts the size of the array must be known.

```
S216 STRUCTURE/UNION cannot be used as an I/O item
```

```
S217 ENCODE/DECODE buffer must be a variable, array, or array element
```

```
S218 Statement labeled $ $
```

```
S219 <reserved message number>
```

```
S220 Redefining predefined macro $
```

```
S221 #elif after #else
```

A preprocessor #elif directive was found after a #else directive; only #endif is allowed in this context.

```
S222 #else after #else
```

A preprocessor #else directive was found after a #else directive; only #endif is allowed in this context.

```
S223 #if-directives too deeply nested
```

Preprocessor #if directive nesting exceeded the maximum allowed (currently 10).

```
S224 Actual parameters too long for $
```

The total length of the parameters in a macro call to the indicated macro exceeded the maximum allowed (currently 2048).

```
W225 Argument mismatch for $
```

The number of arguments supplied in the call to the indicated macro did not agree with the number of parameters in the macro's definition.

```
F226 Can't find include file $
```

The indicated include file could not be opened.

```
S227 Definition too long for $
```

The length of the macro definition of the indicated macro exceeded the maximum allowed (currently 2048).

```
S228 EOF in comment
```

The end of a file was encountered while processing a comment.

```
S229 EOF in macro call to $
```

The end of a file was encountered while processing a call to the indicated macro.

```
S230 EOF in string
```

The end of a file was encountered while processing a quoted string.

```
S231 Formal parameters too long for $
```

The total length of the parameters in the definition of the indicated macro exceeded the maximum allowed (currently 2048).

```
S232 Identifier too long
```

The length of an identifier exceeded the maximum allowed (currently 2048).

```
S233 <reserved message number>
```

```
W234 Illegal directive name
```

The sequence of characters following a # sign was not an identifier.

```
W235 Illegal macro name
```

A macro name was not an identifier.

```
S236 Illegal number $
```

The indicated number contained a syntax error.

```
F237 Line too long
```

The input source line length exceeded the maximum allowed (currently 2048).

```
W238 Missing #endif
```

End of file was encountered before a required #endif directive was found.

```
W239 Missing argument list for $
```

A call of the indicated macro had no argument list.

```
S240 Number too long
```

The length of a number exceeded the maximum allowed (currently 2048).

```
W241 Redefinition of symbol $
```

The indicated macro name was redefined.

```
I242 Redundant definition for symbol $
```

A definition for the indicated macro name was found that was the same as a previous definition.

```
F243 String too long
```

The length of a quoted string exceeded the maximum allowed (currently 2048).

```
S244 Syntax error in #define, formal $ not identifier
```

A formal parameter that was not an identifier was used in a macro definition.

```
W245 Syntax error in #define, missing blank after name or arglist
```

There was no space or tab between a macro name or argument list and the macro's definition.

```
S246 Syntax error in #if
```

A syntax error was found while parsing the expression following a #if or #elif directive.

```
S247 Syntax error in #include
```

The #include directive was not correctly formed.

```
W248 Syntax error in #line
```

A #line directive was not correctly formed.

```
W249 Syntax error in #module
```

A #module directive was not correctly formed.

```
W250 Syntax error in #undef
```

A #undef directive was not correctly formed.

```
W251 Token after #ifdef must be identifier
```

The #ifdef directive was not followed by an identifier.

```
W252 Token after #ifndef must be identifier
```

The #ifndef directive was not followed by an identifier.

```
S253 Too many actual parameters to $
```

The number of actual arguments to the indicated macro exceeded the maximum allowed (currently 31).

```
S254 Too many formal parameters to $
```

The number of formal arguments to the indicated macro exceeded the maximum allowed (currently 31).

```
F255 Too much pushback
```

The preprocessor ran out of space while processing a macro expansion. The macro may be recursive.

```
W256 Undefined directive $
```

The identifier following a # was not a directive name.

```
F257 POS value must be positive.
```

A value for POS <= 0 was encountered. Negative and 0 values are illegal for a position in a file.

```
S257 EOF in #include directive
```

End of file was encountered while processing a #include directive.

```
S258 Unmatched #elif
```

A #elif directive was encountered with no preceding #if or #elif directive.

```
S259 Unmatched #else
```

A #else directive was encountered with no preceding #if or #elif directive.

```
S260 Unmatched #endif
```

A #endif directive was encountered with no preceding #if, #ifdef, or #ifndef directive.

```
S261 Include files nested too deeply
```

The nesting depth of #include directives exceeded the maximum (currently 20).

```
S262 Unterminated macro definition for $
```

A newline was encountered in the formal parameter list for the indicated macro.

```
S263 Unterminated string or character constant
```

A newline with no preceding backslash was found in a quoted string.

```
I264 Possible nested comment
```

The characters /* were found within a comment.

```
S265 <reserved message number>
```

```
S266 <reserved message number>
```

```
S267 <reserved message number>
```

```
W268 Cannot inline subprogram; common block mismatch
```

```
W269 Cannot inline subprogram; argument type mismatch
```

This message may be severe if the compilation has gone too far to undo the inlining process.

```
F270 Missing -exlib option
```

```
W271 Can't inline $ - wrong number of arguments
```

```
I272 Argument of inlined function not used
```

```
S273 Inline library not specified on command line (-inlib switch)
```

```
F274 Unable to access file $/TOC
```

```
S275 Unable to open file $ while extracting or inlining
```

```
F276 Assignment to constant actual parameter in inlined subprogram
```

```
I277 Inlining of function $ may result in recursion
```

```
S278 <reserved message number>
```

```
W279 Possible use of $ before definition in $
```

The optimizer has detected the possibility that a variable is used before it has been assigned a value. The names of the variable and the function in which the use occurred are listed. The line number, if specified, is the line number of the basic block containing the use of the variable.

```
W280 Syntax error in directive $
```

Messages 280-300 reserved for directives handling

```
W281 Directive ignored - $ $
```

```
S300 Too few data constants in initialization of derived type $
```

```
S301 $ must be TEMPLATE or PROCESSOR
```

```
S302 Unmatched END$ statement
```

```
S303 END statement for $ required in an interface block
```

```
S304 EXIT/CYCLE statement must appear in a DO/DOWHILE loop$
```

```
S305 $ cannot be named, $
```

```
S306 $ names more than one construct
```

```
S307 $ must have the construct name $
```

```
S308 DO may not terminate at an EXIT, CYCLE, RETURN, STOP, GOTO, or arithmetic IF
```

```
S309 Incorrect name, $, specified in END statement
```

```
S310 $ $
```

Generic message for MODULE errors.

```
W311 Non-replicated mapping for $ array, $, ignored
```

```
W312 <reserved message number>
```

```
W313 <reserved message number>
```

```
E314 IPA: actual argument $ is a label, but dummy argument $ is not an asterisk
```

The call passes a label to the subprogram; the corresponding dummy argument in the subprogram should be an asterisk to declare this as the alternate return.

```
I315 IPA: routine $, $ constant dummy arguments
```

This many dummy arguments are being replaced by constants due to interprocedural analysis.

```
I316 IPA: routine $, $ INTENT(IN) dummy arguments
```

This many dummy arguments are being marked as INTENT(IN) due to interprocedural analysis.

```
I317 <reserved message number>
```

```
I318 <reserved message number>
```

```
I319 <reserved message number>
```

```
I320 IPA: routine $, $ common blocks optimized
```

This many mapped common blocks were optimized by interprocedural analysis.

```
I321 IPA: routine $, $ common blocks not optimized
```

This many mapped common blocks were not optimized by interprocedural analysis, either because they were declared differently in different routines, or they did not appear in the main program.

```
I322 IPA: analyzing main program $
```

Interprocedural analysis is building the call graph and propagating information with the named main program.

```
I323 IPA: collecting information for $
```

Interprocedural analysis is saving information for the current subprogram for subsequent analysis and propagation.

```
W324 IPA file $ appears to be out of date
```

```
W325 IPA file $ is for wrong subprogram: $
```

```
W326 Unable to open file $ to propagate IPA information to $
```

```
I327 IPA: $ subprograms analyzed
```

```
I328 IPA: $ dummy arguments replaced by constants
```

```
I329 IPA: $ INTENT(IN) dummy arguments should be INTENT(INOUT)
```

```
I330 IPA: $ dummy arguments changed to INTENT(IN)
```

```
I331 <reserved message number>
```

```
I332 <reserved message number>
```

```
I333 <reserved message number>
```

```
I334 <reserved message number>
```

```
I335 <reserved message number>
```

```
I336 <reserved message number>
```

```
I337 IPA: $ common blocks optimized
```

```
I338 IPA: $ common blocks not optimized
```

```
S339 Bad IPA contents file: $
```

```
S340 Bad IPA file format: $
```

```
S341 Unable to create file $ while analyzing IPA information
```

```
S342 Unable to open file $ while analyzing IPA information
```

```
S343 Unable to open IPA contents file $
```

```
S344 Unable to create file $ while collecting IPA information
```

```
F345 Internal error in $: table overflow
```

Analysis failed due to a table overflowing its maximum size.

```
W346 Subprogram $ appears twice
```

The subprogram appears twice in the same source file; IPA will ignore the first appearance.

```
F347 Missing -ipalib option
```

Interprocedural analysis, enabled with the `-ipacollect`, `-ipaanalyze`, or `-ipapropagate` options, requires the `-ipalib` option to specify the library directory.

```
W348 <reserved message number>
```

```
W349 <reserved message number>
```

```
W350 <reserved message number>
```

```
W351 Wrong number of arguments passed to $
```

The subroutine or function statement for the given subprogram has a different number of dummy arguments than appear in the call.

```
W352 Wrong number of arguments passed to $ when bound to $
```

The subroutine or function statement for the given subprogram has a different number of dummy arguments than appear in the call to the EXTERNAL name given.

```
W353 Subprogram $ is missing
```

A call to a subroutine or function with this name appears, but it could not be found or analyzed.

```
I354 Subprogram $ is not called
```

No calls to the given subroutine or function appear anywhere in the program.

```
W355 Missing argument in call to $
```

A nonoptional argument is missing in a call to the given subprogram.

```
I356 Array section analysis incomplete
```

Interprocedural analysis for array section arguments is incomplete; some information may not be available for optimization.

```
I357 Expression analysis incomplete
```

Interprocedural analysis for expression arguments is incomplete; some information may not be available for optimization.

```
W358 Dummy argument $ is EXTERNAL, but actual is not subprogram
```

The call statement passes a scalar or array to a dummy argument that is declared EXTERNAL.

```
W359 SUBROUTINE $ passed to FUNCTION dummy argument $
```

The call statement passes a subroutine name to a dummy argument that is used as a function.

```
W360 FUNCTION $ passed to FUNCTION dummy argument $ with different result type
```

The call statement passes a function argument to a function dummy argument, but the dummy has a different result type.

```
W361 FUNCTION $ passed to SUBROUTINE dummy argument $
```

The call statement passes a function name to a dummy argument that is used as a subroutine.

```
W362 Argument $ has a different type than dummy argument $
```

The type of the actual argument is different than the type of the corresponding dummy argument.

```
W363 Dummy argument $ is a POINTER but actual argument $ is not
```

The dummy argument is a pointer, so the actual argument must be also.

```
W364 Array or array expression passed to scalar dummy argument $
```

The actual argument is an array, but the dummy argument is a scalar variable.

```
W365 Scalar or scalar expression passed to array dummy argument $
```

The actual argument is a scalar variable, but the dummy argument is an array.

```
F366 Internal error: interprocedural analysis fails
```

An internal error occurred during interprocedural analysis; please report this to the compiler maintenance group. If user errors were reported when collecting IPA information or during IPA analysis, correcting them may avoid this error.

```
I367 Array $ bounds cannot be matched to formal argument
```

Passing a nonsequential array to a sequential dummy argument may require copying the array to sequential storage. The most common cause is passing an ALLOCATABLE array or array expression to a dummy argument that is declared with explicit bounds. Declaring the dummy argument as assumed shape, with bounds (:,:,:), will remove this warning.

```
W368 Array-valued expression passed to scalar dummy argument $
```

The actual argument is an array-valued expression, but the dummy argument is a scalar variable.

```
W369 Dummy argument $ has different rank than actual argument
```

The actual argument is an array or array-valued expression with a different rank than the dummy argument.

```
W370 Dummy argument $ has different shape than actual argument
```

The actual argument is an array or array-valued expression with a different shape than the dummy argument; this may require copying the actual argument into sequential storage.

```
W371 Dummy argument $ is INTENT(IN) but may be modified
```

The dummy argument was declared as INTENT(IN), but analysis has found that the argument may be modified; the INTENT(IN) declaration should be changed.

```
W372 <reserved message number>
```

```
I373 <reserved message number>
```

```
I374 <reserved message number>
```

```
I375 <reserved message number>
```

```
I376 <reserved message number>
```

```
I377 <reserved message number>
```

```
I378 <reserved message number>
```

```
I379 <reserved message number>
```

```
I380 <reserved message number>
```

```
I381 <reserved message number>
```

```
I382 IPA: $ subprograms analyzed
```

Interprocedural analysis succeeded in finding and analyzing this many subprograms in the whole program.

```
I383 IPA: $ dummy arguments replaced by constants
```

Interprocedural analysis has found this many dummy arguments in the whole program that can be replaced by constants.

```
I384 IPA: $ dummy arguments changed to INTENT(IN)
```

Interprocedural analysis has found this many dummy arguments in the whole program that are not modified and can be declared as INTENT(IN).

```
W385 IPA: $ INTENT(IN) dummy arguments should be INTENT(INOUT)
```

Interprocedural analysis has found this many dummy arguments in the whole program that were declared as INTENT(IN) but should be INTENT(INOUT).

```
I386 <reserved message number>
```

```
I387 <reserved message number>
```

```
I388 <reserved message number>
```

```
I389 <reserved message number>
```

```
I390 <reserved message number>
```

```
I391 <reserved message number>
```

```
I392 IPA: $ common blocks optimized
```

Interprocedural analysis has found this many common blocks that could be optimized.

```
I393 IPA: $ common blocks not optimized
```

Interprocedural analysis has found this many common blocks that could not be optimized, either because the common block was not declared in the main program, or because it was declared differently in different subprograms.

```
I394 IPA: $ replaced by constant value
```

The dummy argument was replaced by a constant as per interprocedural analysis.

```
I395 IPA: $ changed to INTENT(IN)
```

The dummy argument was changed to INTENT(IN) as per interprocedural analysis.

```
I396 <reserved message number>
```

```
I397 <reserved message number>
```

```
I398 <reserved message number>
```

```
I399 IPA: common block $ not optimized
```

The given common block was not optimized by interprocedural analysis either because it was not declared in the main program, or because it was declared differently in different subprograms.

```
E400 IPA: dummy argument $ is an asterisk, but actual argument is not a label
```

The subprogram expects an alternate return label for this argument.

```
E401 Actual argument $ is a subprogram, but Dummy argument $ is not declared EXTERNAL
```

The call statement passes a function or subroutine name to a dummy argument that is a scalar variable or array.

```
E402 Actual argument $ is illegal
```

```
E403 <reserved message number>
```

```
E404 <reserved message number>
```

```
E405 <reserved message number>
```

```
E406 <reserved message number>
```

```
W407 Argument $ has a different character length than dummy argument $
```

The character length of the actual argument is different than the length specified for the corresponding dummy argument.

```
W408 Specified main program $ is not a PROGRAM
```

The main program specified on the command line is a subroutine, function, or block data subprogram.

```
W409 More than one main program in IPA directory: $ and $
```

There is more than one main program analyzed in the IPA directory shown. The first one found is used.

```
W410 No main program found; IPA analysis fails.
```

The main program must appear in the IPA directory for analysis to proceed.

```
W411 Formal argument $ is DYNAMIC but actual argument is an expression
```

```
W412 Formal argument $ is DYNAMIC but actual argument $ is not
```

```
I413 Formal argument $ has two reaching distributions and may be a candidate for
↪cloning
```

```
I414 $ and $ may be aliased and one of them is assigned
```

Interprocedural analysis has determined that two formal arguments may be aliased because the same variable is passed in both argument positions; or one formal argument and a global or COMMON variable may be aliased, because the global or COMMON variable is passed as an actual argument. If either alias is assigned in the subroutine, unexpected results may occur; this message alerts the user that this situation is disallowed by the Fortran standard.

```
F415 IPA fails: incorrect IPA file
```

Interprocedural analysis saves its information in special IPA files in the specified IPA directory. One of these files has been renamed or corrupted. This can arise when there are two files with the same prefix, such as `a.hpf` and `a.f90`.

```
E416 Argument $ has the SEQUENCE attribute, but the dummy parameter $ does not
```

When an actual argument is an array with the SEQUENCE attribute, the dummy parameter must have the SEQUENCE attribute or an INTERFACE block must be used.

```
E417 Interface block for $ is a SUBROUTINE but should be a FUNCTION
```

```
E418 Interface block for $ is a FUNCTION but should be a SUBROUTINE
```

```
E419 Interface block for $ is a FUNCTION has wrong result type
```

```
W420 Earlier $ directive overrides $ directive
```

```
W421 $ directive can only appear in a function or subroutine
```

```
E422 Nonconstant DIM= argument is not supported
```

```
E423 Constant DIM= argument is out of range
```

```
E424 Equivalence using substring or vector triplets is not allowed
```

```
E425 A record is not allowed in this context
```

```
E426 WORD type cannot be converted
```

```
E427 Interface block for $ has wrong number of arguments
```

```
E428 Interface block for $ should have $
```

```
E429 Interface block for $ should not have $
```

```
E430 Interface block for $ has wrong $
```

```
W431 Program is too large for Interprocedural Analysis to complete
```

```
W432 Illegal type conversion $
```

```
E433 <reserved message number>
```

```
W434 Incorrect home array specification ignored
```

```
W435 Array declared with zero size
```

An array was declared with a zero or negative dimension bound, as ‘real a(-1)’, or an upper bound less than the lower bound, as ‘real a(4:2)’.

```
W436 Independent loop not parallelized$
```

```
W437 Type $ will be mapped to $
```

Where DOUBLE PRECISION is not supported, it is mapped to REAL, and similarly for COMPLEX(16) or COMPLEX*32.

```
E438 $ $ not supported on this platform
```

This construct is not supported by the compiler for this target.

```
S439 An internal subprogram cannot be passed as argument - $
```

```
S440 Defined assignment statements may not appear in WHERE statement or WHERE block
```

```
S441 $ may not appear in a FORALL block
```

```
E442 Adjustable-length character type not supported on this host - $ $
```

```
S443 EQUIVALENCE of derived types not supported on this host - $
```

```
S444 Derived type in EQUIVALENCE statement must have SEQUENCE attribute - $
```

A variable or array with derived type appears in an EQUIVALENCE statement. The derived type must have the SEQUENCE attribute, but does not.

```
E445 Array bounds must be integer $ $
```

The expressions in the array bounds must be integer.

```
S446 Argument number $ to $: rank mismatch
```

The number of dimensions in the array or array expression does not match the number of dimensions in the dummy argument.

```
S447 Argument number $ to $ must be a subroutine or function name
```

```
S448 Argument number $ to $ must be a subroutine name
```

```
S449 Argument number $ to $ must be a function name
```

```
S450 Argument number $ to $: kind mismatch
```

```
S451 Arrays of derived type with a distributed member are not supported
```

```
S452 Assumed length character, $, is not a dummy argument
```

```
S453 Derived type variable with pointer member not allowed in IO - $ $
```

```
S454 Subprogram $ is not a module procedure
```

Only names of module procedures declared in this module or accessed through USE association can appear in a MODULE PROCEDURE statement.

```
S455 A derived type array section cannot appear with a member array section - $
```

A reference like A(:)%B(:), where ‘A’ is a derived type array and ‘B’ is a member array, is not allowed; a section subscript may appear after ‘A’ or after ‘B’, but not both.

```
S456 Unimplemented for data type for MATMUL
```

```
S457 Illegal expression in initialization
```

```
S458 Argument to NULL() must be a pointer
```

```
S459 Target of NULL() assignment must be a pointer
```

```
S460 ELEMENTAL procedures cannot be RECURSIVE
```

```
S461 Dummy arguments of ELEMENTAL procedures must be scalar
```

```
S462 Arguments and return values of ELEMENTAL procedures cannot have the POINTER
↪attribute
```

```
S463 Arguments of ELEMENTAL procedures cannot be procedures
```

```
S464 An ELEMENTAL procedure cannot be passed as argument - $
```

```
S465 Functions returning a POINTER require an explicit interface
```

```
S466 Member $ of derived type $ has PRIVATE type
```

```
S467 Target of NULL() assignment must have the ALLOCATABLE attribute
```

```
W468 Argument to ISO_C_BINDING intrinsic must have TARGET attribute set
```

```
W469 Character argument to C_LOC intrinsic must have length of one
```

```
W470 <reserved message number>
```

```
W471 <reserved message number>
```

```
E472 A Scalar element of a nonsequential array cannot be passed to a dummy array
↪argument - $
```

A subroutine or function call may not pass an element of an array, like 'A(N)', to a dummy array argument if the array 'A' is not sequential. If the array is sequential, then Fortran sequence and storage association rules will treat the dummy argument as a new array equivalenced to the actual argument starting at the element passed. If the array is not sequential, then Fortran sequence and storage association rules do not apply.

```
W473 $ must have the PURE attribute
```

```
F474 <reserved message number>
```

```
E475 <reserved message number>
```

```
E476 <reserved message number>
```

```
E477 The device array section actual argument was not stride-1 in the leading
↪dimension - $
```

A device (device, shared, or constant attribute) array passed as an array section to an assumed-shape dummy argument must be stride-1 in the leading dimension.

```
E478 Invalid actual argument to REFLECTED dummy argument - $
```

The actual argument symbol or expression to a dummy argument with the Accelerator REFLECTED attribute must be a symbol that has a visible device copy. Expressions are not allowed.

```
E479 The dummy argument $ is REFLECTED; the actual argument $ must have a visible
↪device copy
```

If a dummy argument has the Accelerator REFLECTED attribute, the actual argument must be a symbol with a visible device copy. This may be because the symbol appeared in a MIRROR, REFLECTED, COPYIN, COPYOUT, COPY or LOCAL declarative Accelerator directive, or because it appeared in a COPYIN, COPYOUT, COPY or LOCAL clause for an Accelerator DATA REGION or REGION surrounding the procedure call.

```
E480 Argument $ is passed to dummy argument $, which is REFLECTED; the actual
↪argument must not require runtime reshaping
```

When an actual argument is an array section or pointer array section, sometimes the actual argument must be copied to a temporary array. This may occur if the dummy argument is not assumed-shape, and so must be contiguous in memory, or if the actual argument is not stride-1 in the leftmost (first) dimension. In these cases, the REFLECTED argument is not supported.

```
F481 An ENTRY name must not appear as a dummy argument - $
```

The name of the subprogram or an ENTRY to the subprogram must not appear as a dummy argument to the subprogram.

```
E482 <reserved message number>
```

```
E483 <reserved message number>
```

```
E484 <reserved message number>
```

```
E485 <reserved message number>
```

```
E486 The dummy argument $ is REFLECTED; an array element cannot be passed to a
↪REFLECTED argument
```

An actual argument that is an array element cannot be passed to a REFLECTED dummy argument.

```
E487 Index variable $ does not appear in a subscript on the left hand side of the
↪FORALL assignment
```

In a FORALL statement, each index variable in the FORALL must appear in some subscript of the left hand side of the FORALL assignment. Otherwise, the FORALL will assign the same left hand side elements for different values of that index.

```
I489 <reserved message number>
```

```
E488 The function call in the FORALL does not have the PURE attribute - $
```

In a FORALL statement, all functions used must be PURE or ELEMENTAL. Otherwise, they cannot be called in parallel.

```
E490 An array section of $ is passed to the REFLECTED argument $, which is not
↪supported
```

When an actual argument is an array section, the dummy argument must not have the REFLECTED attribute.

```
W491 <reserved message number>
```

```
E492 <reserved message number>
```

```
E493 <reserved message number>
```

```
E494 <reserved message number>
```

```
W495 <reserved message number>
```

```
I496 <reserved message number>
```

```
E497 <reserved message number>
```

```
E498 <reserved message number>
```

```
W499 <reserved message number>
```

```
E500 MODULE $ uses (directly or indirectly) MODULE $, which causes a USE cycle
```

If MODULE A has a USE statement for MODULE B, we say that MODULE A directly uses MODULE B. If MODULE B has a USE statement for MODULE C, we say that MODULE A indirectly uses MODULE C. If MODULE C then has a USE statement for MODULE A, then MODULE A indirectly uses itself, which is a USE cycle, and is not allowed.

```
E504 DIM argument out of range for this symbol - $
```

The DIM argument to this transformation intrinsic (CSHIFT, EOSHIFT, ...) must be between 1 and the rank of the array or expression being transformed.

```
E505 DIM argument out of range for this reduction - $
```

The DIM argument to this reduction intrinsic (SUM, PRODUCT, ...) must be between 1 and the rank of the expression being reduced.

```
E506 The argument to ASSOCIATED must be a pointer - $
```

The argument to the ASSOCIATED intrinsic function must be a variable or array with the POINTER attribute.

```
E507 The arguments to MOVE_ALLOC must be ALLOCATABLE - $
```

The arguments to the MOVE_ALLOC procedure must have the ALLOCATABLE attribute.

```
E508 The array objects in a call to an elemental function are not conformable - $
```

When calling an elemental function, the arguments must be scalars or conformable arrays or array expressions.

```
E509 Variables in a PURE subprogram may not have the SAVE attribute - $
```

PURE subprograms cannot refer to external, module, or COMMON data, and cannot save state in a SAVEd variable.

```
E510 Only assignment statements are allowed in a WHERE construct
```

A WHERE construct is the WHERE statement and all the statements until the matching ENDWHERE. The body of the WHERE construct can only contain assignment statements.

```
E511 The WHERE mask expression and the array assignment do not conform
```

The assignment under control of a WHERE mask must have the same shape as the WHERE mask.

```
E512 The WHERE mask is not an array expression
```

The WHERE mask expression must be a logical array expression.

```
E513 <reserved message number>
```

```
E514 <reserved message number>
```

```
E515 <reserved message number>
```

```
E516 <reserved message number>
```

```
E517 <reserved message number>
```

```
W518 <reserved message number>
```

```
E519 More than one device-resident object in assignment
```

Only one device-resident variable or array is allowed in an assignment.

```
E520 Host MODULE data cannot be used in a DEVICE or GLOBAL subprogram - $
```

CUDA Fortran DEVICE or GLOBAL subprograms cannot access host data directly.

```
E521 MODULE data cannot be used in a DEVICE or GLOBAL subprogram unless compiling for
↪compute capability >= 2.0 - $
```

CUDA Fortran DEVICE or GLOBAL subprograms cannot access data from any MODULE except the MODULE containing the subprogram, unless they are being compiled for compute capability 2.0 or higher. This feature requires the unified memory system provided in compute capability 2.0.

```
E522 MODULE data cannot be used in a DEVICE or GLOBAL subprogram unless compiling
↪with CUDA Toolkit 3.0 or later - $
```

CUDA Fortran DEVICE or GLOBAL subprograms cannot access data from any MODULE except the MODULE containing the subprogram, unless they are being compiled for compute capability 2.0 or higher with the CUDA Toolkit 3.0 or later.

This feature requires the unified memory system provided in compute capability 2.0.

```
W523 MODULE data used in a DEVICE or GLOBAL subprogram forces compute capability >= 2.
↪0 only - $
```

CUDA Fortran DEVICE or GLOBAL subprograms can access MODULE data only when compiled for compute capability 2.0 or greater.

```
E524 Dependency in assignment causes allocation of a temporary which is not supported
↪in DEVICE or GLOBAL subprograms
```

The compiler has identified a possible dependency in an assignment statement which requires allocation of temporary storage to produce a correct result. Dynamic allocation of memory is not supported in subprograms that run on the device.

```
E525 Array reshaping is not supported for device subprogram calls: argument $ to
↪subprogram $
```

Passing an array section or assumed-shape array to a non-assumed-shape dummy argument is not supported in global or device subprograms. This would require a run-time test and a possible run-time copy to a dynamically allocated temporary array.

```
W526 SHARED attribute ignored on dummy argument $
```

The SHARED attribute has no meaning when applied to a dummy argument.

```
E527 Argument number $ requires allocation of a temporary which is not supported in
↪DEVICE or GLOBAL subprograms
```

Evaluation of the specified argument requires allocation of temporary storage for the result to be passed to the subprogram being called. Dynamic allocation of memory is not supported in subprograms that run on the device.

```
E528 Argument number $ to $: device attribute mismatch
```

Device attributes of the actual and formal arguments are not the same.

```
E529 PRINT and WRITE statements in device subprograms are only supported when
↪compiling with CUDA Toolkit 4.0 or later
```

Support for PRINT * or WRITE(*,*) statements in CUDA Fortran device subprograms requires CUDA Toolkit 4.0 or later and compute capability 2.0 or higher.

```
E530 PRINT and WRITE statements in device subprograms are only supported with compute
↪capability 2.0 or higher
```

Support for PRINT * or WRITE(*,*) statements in CUDA Fortran device subprograms requires CUDA Toolkit 4.0 or later and compute capability 2.0 or higher.

```
W531 NVIDIA extension to OpenACC: $
```

This program is using an NVIDIA extension to OpenACC.

```
W532 OpenACC feature not yet implemented: $
```

This OpenACC feature is not yet implemented. This program is using an NVIDIA extension to OpenACC.

```
E533 Clause $ not allowed in $ directive
```

This clause is not allowed on the specified directive.

```
E534 A loop scheduling directive may not appear within a KERNEL loop
```

An accelerator or OpenACC loop directive that specifies a schedule, such as PARALLEL, VECTOR, WORKER or GANG, may not appear inside a loop that has an accelerator loop directive with the KERNEL clause. This clause is not allowed on the specified directive.

```
E535 Undeclared symbol $ used in directive
```

Symbols used in OpenACC directives must be declared.

```
S901 #elif after #else
```

A preprocessor #elif directive was found after a #else directive; only #endif is allowed in this context.

```
S902 #else after #else
```

A preprocessor #else directive was found after a #else directive; only #endif is allowed in this context.

```
W905 Argument mismatch for $
```

The number of arguments supplied in the call to the indicated macro did not agree with the number of parameters in the macro's definition.

```
F906 Can't find include file $
```

The indicated include file could not be opened.

```
S908 EOFin comment
```

The end of a file was encountered while processing a comment.

```
S909 EOFin macro call to $
```

The end of a file was encountered while processing a call to the indicated macro.

```
S912 Identifier too long
```

The length of an identifier exceeded the maximum allowed (currently 2048).

```
W914 Illegal directive name
```

The sequence of characters following a # sign was not an identifier.

```
W915 Illegal macro name
```

A macro name was not an identifier.

```
W918 Missing #endif
```

End of file was encountered before a required #endif directive was found.

```
W919 Missing argument list for $
```

A call of the indicated macro had no argument list.

```
S920 Number too long
```

The length of a number exceeded the maximum allowed (currently 2048).

```
W921 Redefinition of symbol $
```

The indicated macro name was redefined.

```
I922 Redundant definition for symbol $
```

A definition for the indicated macro name was found that was the same as a previous definition.

```
F923 String too long
```

The length of a quoted string exceeded the maximum allowed (currently 2048).

```
S924 Syntax error in #define, formal $ not identifier
```

A formal parameter that was not an identifier was used in a macro definition.

```
S926 Syntax error in #if
```

A syntax error was found while parsing the expression following a #if or #elif directive.

```
S927 Syntax error in #include
```

The #include directive was not correctly formed.

```
W928 Syntax error in #line
```

A #line directive was not correctly formed.

```
W929 Syntax error in #module
```

A #module directive was not correctly formed.

```
W930 Syntax error in #undef
```

A #undef directive was not correctly formed.

```
W931 Token after #ifdef must be identifier
```

The #ifdef directive was not followed by an identifier.

```
W932 Token after #ifndef must be identifier
```

The #ifndef directive was not followed by an identifier.

```
S933 Too many actual parameters to $
```

The number of actual arguments to the indicated macro exceeded the maximum allowed (currently 31).

```
S934 Too many formal parameters to $
```

The number of formal arguments to the indicated macro exceeded the maximum allowed (currently 31).

```
S935 Illegal context for __VA_ARGS__
```

```
W936 Undefined directive $
```

The identifier following a # was not a directive name.

```
S937 EOFin #include directive
```

End of file was encountered while processing a #include directive.

```
S938 Unmatched #elif
```

A #elif directive was encountered with no preceding #if or #elif directive.

```
S939 Unmatched #else
```

A #else directive was encountered with no preceding #if or #elif directive.

```
S940 Unmatched #endif
```

A #endif directive was encountered with no preceding #if, #ifdef, or #ifndef directive.

```
W941 Illegal token in directive, $
```

A directive token contains a illegal character.

```
S942 Unterminated macro definition for $
```

A newline was encountered in the formal parameter list for the indicated macro.

```
S943 Unterminated string or character constant
```

A newline with no preceding backslash was found in a quoted string.

```
I944 Possible nested comment
```

The characters /* were found within a comment.

```
I945 Redefining predefined macro $
```

```
I946 Undefining predefined macro $
```

```
W947 Can't redefine predefined macro $
```

```
W948 Can't undefine predefined macro $
```

```
F949 #error -- $
```

User defined preprocessor error message.

```
W950 #ident not followed by quoted  string
```

```
W951 Extraneous tokens ignored following # directive
```

```
F952 Unexpected EOF following #directive
```

```
W953 Unexpected # ignored in #if expression
```

```
S954 Illegal number in directive
```

```
S955 Illegal token in #if expression
```

```
S956 Missing > in #include
```

```
W957 Arguments in macro $ are not unique
```

```
S959 ## directive occurs at beginning or end of macro definition
```

```
S960 $ is not an argument
```

```
W961 No macro replacement within a character constant
```

```
W962 Macro replacement within a character constant
```

```
W964 Macro replacement within a string literal
```

```
F965 Recursive include file $
```

```
W966 Null argument to macro
```

Argument to macro is a null value.

```
F967 #warning -- $
```

User defined preprocessor warning message.

```
S969 _Pragma $
```

Pragma operator errors.

```
W972 The directive !$acc mirror is deprecated; use !$acc declare create instead
```

```
W973 The directive !$acc reflected is deprecated; use !$acc declare present
```

```
W974 The directive !$acc region is deprecated; use !$acc kernels instead
```

```
W975 The directive !$acc data region is deprecated; use !$acc data instead
```

```
W976 The directive !$acc do is deprecated; use !$acc loop instead
```

```
W977 The directive !$acc do kernel is deprecated; use !$acc loop instead
```

```
W978 The directive !$acc loop parallel is deprecated; use !$acc loop gang instead
```

```
W979 The directive !$acc region do is deprecated; use !$acc kernels loop instead
```

```
W980 The directive !$acc region loop is deprecated; use !$acc kernels loop instead
```

```
W981 The directive !$acc kernels do is deprecated; use !$acc kernels loop instead
```

```
W982 <reserved message number>

W983 The directive !$acc parallel do is deprecated; use !$acc parallel loop instead

W984 The directive !$acc scalar region is deprecated; use !$acc serial instead

W985 The clause local is deprecated; use clause create instead

W986 The clause cache is deprecated; use directive !$acc cache instead

W987 The clause update host is deprecated; use separate update host directive after
↪the region instead

W988 The clause update device is deprecated; use separate update device directive
↪before the region instead

W989 The clause update in is deprecated; use separate update device directeve before
↪the region instead

W990 The clause update out is deprecated; use update self instead

W991 The clause pnot is deprecated; use no_create instead

W992 The clause updatein is deprecated; use update device instead

W993 The clause updateout is deprecated; use update self instead

W994 The directive !$acc copy is deprecated; use !$acc declare copy instead

W995 The directive !$acc copyin is deprecated; use !$acc declare copyin instead

W996 The directive !$acc copyout is deprecated; use !$acc declare copyout instead

W997 The directive !$acc device_resident is deprecated; use !$acc declare device_
↪resident instead

W998 The directive !$acc do host is deprecated; no OpenACC equivalent

W999 The directive !$acc loop kernel is deprecated; no OpenACC equivalent

S1000 Call in OpenACC region to procedure '$' which has no acc routine information

S1001 All selected compute capabilities were disabled (see -Minfo)

S1002 Reduction type not supported for this variable datatype - $

W1003 Lambda capture by reference not supported in Accellerated region

W1004 Lambda capture 'this' by reference not supported in Accellerated region
```

```
W1005 The clause unroll is deprecated; no OpenACC equivalent

W1006 The clause mirror is deprecated; no OpenACC equivalent

W1007 The clause host is deprecated; no OpenACC equivalent

S1011 Device variable cannot be THREADPRIVATE - $

S1012 Threadprivate variables are not supported in acc routine - $

S1013 Static Threadprivate variables are not supported - $

S1014 Global Threadprivate variables are not supported - $

F1015 No shape directive is defined in structure $

F1016 No shape name $ is defined in structure $

F1017 arrays/pointers appearing in the OpenACC shape and policy directives must be a
 ↪member of current aggregate type

F1018 Only one unnamed Shape directive is allowed in one aggregate type (struct/union)

F1019 Type clause must be used to specified structure type when Shape/Policy is
 ↪defined outside (struct/union/class) definition

F1020 Data-Type appearing in type clause cannot be found

F1021 Data-Type appearing in type clause must be struct/union type

F1022 Duplicated shape names $ are defined for structure/union/class $

F1023 Duplicated policy names $ are defined for structure/union/class $

F1024 Type clause is not allowed within structure/union/class definition

F1025 The number of dimension section descriptions doesn't match member $ which
 ↪requires $ dimensions

F1026 Pointers appearing within relative clause must be their sibling members

F1027 As motion clauses, only create, copyin, copyout, copy, update, and deviceptr
 ↪are allowed in policy directive

F1028 The variable $ doesn't have predefined policy $ available

F1029 The variable $ using policy $ is not a structure-based type

F1030 Policy motion $ is not allowed in $ directive
```

```
W1031 The directive !$acc create is deprecated; use !$acc declare create instead
```

```
W1032 The directive !$acc present is deprecated; use !$acc declare present instead
```

```
W1033 The directive !$acc link is deprecated; use !$acc declare link instead
```

```
F1034 Only signed/unsigned 32 bits and 64 bits integer variables are allowed in bound
↪expression. $ is is not such variable
```

```
F1035 Only integer sibling members and global variables are allowed in bound
↪expression. $ is is neither of them.
```

```
F1036 No global variable named $ has been defined
```

```
F1037 Default clause can only contain include and exclude keyword.
```

```
F1038 Var $ used in array region cannot be found
```

```
F1039 Var $ used is not an integer type. It has to be int4 and int8.
```

```
F1040 In Fortran, the default option is full deep copy. A shape directive must be
↪given a shape name.
```

```
F1041 Shape and policy directives cannot be declared within routine/subroutine.
```

```
S1042 $ mask expression must be scalar
```

A DO CONCURRENT or FORALL mask expression must be scalar.

```
S1043 DO CONCURRENT $ references construct variable $
```

A DO CONCURRENT limit or step control expression may not reference an index name or LOCAL name. A DO CONCURRENT mask expression may not reference a LOCAL name.

```
S1044 Invalid DO CONCURRENT locality spec variable $
```

A name in a DO CONCURRENT locality spec must be a valid variable name in the containing scope.

```
S1045 DO CONCURRENT index name $ may not appear in a locality spec
```

```
S1046 Variable $ has multiple DO CONCURRENT locality spec references
```

```
S1047 Multiple DO CONCURRENT DEFAULT(NONE) locality specs
```

```
S1048 LOCAL/LOCAL_INIT variable $ $
```

A DO CONCURRENT LOCAL or LOCAL_INIT variable must not have the ALLOCATABLE, INTENT (IN), or OPTIONAL attribute, must not be of finalizable type, must not be a nonpointer polymorphic dummy argument, must not be a an assumed-size array, and must be permitted to appear in a variable definition context.

```
S1049 Variable $ is not in a DO CONCURRENT locality list
```

When DEFAULT(NONE) is specified for a DO CONCURRENT loop, construct variables and variables from containing scopes must appear in a locality spec.

```
S1050 $ DO CONCURRENT construct
```

A DO CONCURRENT construct may not contain a RETURN, EXIT, GOTO, or other branch out of the construct. A CYCLE statement is permitted.

```
S1051 DO CONCURRENT polymorphic variable deallocation - $
```

A DO CONCURRENT construct must not contain a statement that might result in the deallocation of a polymorphic variable.

```
S1052 $ call in DO CONCURRENT construct
```

A DO CONCURRENT construct may not contain a call to IEEE_GET_FLAG, IEEE_SET_HALTING_MODE, or IEEE_GET_HALTING_MODE from intrinsic module IEEE_EXCEPTIONS.

```
S1053 Duplicate $ index name
```

A DO CONCURRENT or FORALL construct or statement may not specify an index name multiple times.

```
W1054 Duplicate subprogram prefix $ is used
```

```
S1055 MODULE prefix cannot be inside an abstract interface
```

```
S1056 MODULE prefix is only allowed for subprograms that were declared as separate
↪module procedures
```

```
S1057 Definition argument name $ does not match declaration argument name $
```

```
S1058 The type of definition argument $ does not match its declaration type
```

```
S1059 The definition of subprogram $ does not have the same number of arguments as
↪its declaration
```

```
S1060 The $ of the definition and declaration of subprogram $ must match
```

```
S1061 The definition of function return type of $ does not match its declaration type
```

```
S1062 LOCAL_INIT variable does not have an outside variable of the same name - $
```

A DO CONCURRENT variable with LOCAL_INIT locality must have a host variable of the same name.

```
W1063 Data construct ignored in compute construct or acc routine
```

```
S1065 Unsupported nested compute construct in compute construct or acc routine
```

```
S1066 The -cuda flag should be used with CUDA DEVICE variable - $
```

```
S1067 Cannot determine bounds for array - $
```

```
S1068 Cannot determine start offset for array - $
```

```
S1069 Data clause required with default(none) - $
```

```
W1070 Data clause required in OpenACC 2.7 with default(none) - $; a future release
↪will enforce this
```

```
S1071 Host array used in CUF kernel - $
```

```
S1100 Cannot collapse non-tightly-nested loops
```

```
S1207 ERROR STOP stop-code requires either a character or integer expression.
```

```
S1208 QUIET requires a logical expression.
```

```
S1209 ERROR STOP stop-code integer expression must be an integer of default kind.
```

```
S1210 Parent module $ must declare a separate module procedure.
```

```
S1211 Submodule's ancestor module $ must be a nonintrinsic module.
```

```
S1212 $ was previously declared to be a module procedure.
```

```
S1213 OpenACC $ data clause may not follow a device_type clause.
```

```
S1214 PGI Accelerator $ data clause may not follow a device_type clause.
```

```
S1215 OpenACC data clause expected after $.
```

```
S1216 Expression in assignment statement contains type bound procedure name $. This
↪may be a function call that's missing parentheses.
```

```
S1217 Left hand side of polymorphic assignment must be allocatable - $
```

```
S1218 $ statement may not appear in a BLOCK construct.
```

```
S1219 Unimplemented feature: $.
```

```
S1220 PUBLIC namelist /$/ has a PRIVATE namelist object ($)
```

```
S1221 Interface $ is not declared.
```

```
S1222 Invalid module. Interface $ referenced in module $ is not declared.
```

# 9.4. Fortran Run-time Error Messages

This section presents the error messages generated by the run-time system. The run-time system displays error messages on standard output.

## 9.4.1. Message Format

The messages are numbered but have no severity indicators because they all terminate program execution.

## 9.4.2. Message List

Here are the run-time error messages:

`201 illegal value for specifier`

An improper specifier value has been passed to an I/O run-time routine. Example: within an OPEN statement, form='unknown'.

`202 conflicting specifiers`

Conflicting specifiers have been passed to an I/O run-time routine. Example: within an OPEN statement, form='unformatted', blank='null'.

`203 record length must be specified`

A recl specifier required for an I/O run-time routine has not been passed. Example: within an OPEN statement, access='direct' has been passed, but the record length has not been specified (recl=specifier).

`204 illegal use of a readonly file`

Self explanatory. Check file and directory modes for readonly status.

`205 'SCRATCH' and 'SAVE'/'KEEP' both specified`

In an OPEN statement, a file disposition conflict has occurred. Example: within an OPEN statement, status='scratch' and dispose='keep' have both been passed.

`206 attempt to open a named file as 'SCRATCH'`

`207 file is already connected to another unit`

`208 'NEW' specified for file that already exists`

`209 'OLD' specified for file that does not exist`

`210 dynamic memory allocation failed`

Memory allocation operations occur only in conjunction with namelist I/O. The most probable cause of fixed buffer overflow is exceeding the maximum number of simultaneously open file units.

`211 invalid file name`

`212 invalid unit number`

A file unit number less than or equal to zero has been specified.

215 formatted/unformatted file conflict

Formatted/unformatted file operation conflict.

217 attempt to read past end of file

219 attempt to read/write past end of record

For direct access, the record to be read/written exceeds the specified record length.

220 write after last internal record

221 syntax error in format string

A run-time encoded format contains a lexical or syntax error.

222 unbalanced parentheses in format string

223 illegal P or T edit descriptor - value missing

224 illegal Hollerith or character string in format

An unknown token type has been found in a format encoded at run-time.

225 lexical error -- unknown token type

226 unrecognized edit descriptor letter in format

An unexpected Fortran edit descriptor (FED) was found in a run-time format item.

228 end of file reached without finding group

229 end of file reached while processing group

230 scale factor out of range -128 to 127

Fortran P edit descriptor scale factor not within range of -128 to 127.

231 error on data conversion

233 too many constants to initialize group item

234 invalid edit descriptor

An invalid edit descriptor has been found in a format statement.

235 edit descriptor does not match item type

Data types specified by I/O list item and corresponding edit descriptor conflict.

236 formatted record longer than 2000 characters

237 quad precision type unsupported

238 tab value out of range

A tab value of less than one has been specified.

239 entity name is not member of group

240 no initial left parenthesis in format string

241 unexpected end of format string

242 illegal operation on direct access file

243 format parentheses nesting depth too great

244 syntax error - entity name expected

245 syntax error within group definition

```
246 infinite format scan for edit descriptor
248 illegal subscript or substring specification
249 error in format - illegal E, F, G or D descriptor
250 error in format - number missing after '.', '-', or '+'
251 illegal character in format string
252 operation attempted after end of file
253 attempt to read non-existent record (direct access)
254 illegal repeat count in format
255 illegal asynchronous I/O operation
256 POS can only be specified for a 'STREAM' file
257 POS value must be positive
258 NEWUNIT requires FILE or STATUS=SCRATCH
```

## Notices

### Notice



### Trademarks

# Copyright