

Arm® Fortran Compiler

Version 20.2

Developer and reference guide



Arm® Fortran Compiler

Developer and reference guide

Copyright © 2018–2020 Arm Limited or its affiliates. All rights reserved.

Release Information

Document History

Issue	Date	Confidentiality	Change
1830-00	20 June 2018	Non-Confidential	Document release for Arm® Fortran Compiler version 18.3
1840-00	17 July 2018	Non-Confidential	Update for Arm® Fortran Compiler version 18.4
1900-00	02 November 2018	Non-Confidential	Update for Arm® Fortran Compiler version 19.0
1910-00	08 March 2019	Non-Confidential	Update for Arm® Fortran Compiler version 19.1
1920-00	07 June 2019	Non-Confidential	Update for Arm® Fortran Compiler version 19.2
1930-00	30 August 2019	Non-Confidential	Update for Arm® Fortran Compiler version 19.3
2000-00	29 November 2019	Non-Confidential	Update for Arm® Fortran Compiler version 20.0
2010-00	23 April 2020	Non-Confidential	Update for Arm® Fortran Compiler version 20.1
2010-01	23 April 2020	Non-Confidential	Documentation update 1 for Arm® Fortran Compiler version 20.1
2020-00	25 June 2020	Non-Confidential	Update for Arm® Fortran Compiler version 20.2

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if

there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2018–2020 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

www.arm.com

Contents

Arm® Fortran Compiler Developer and reference guide

Preface

About this book	11
-----------------------	----

Chapter 1

Get started

1.1 Arm® Fortran Compiler	1-14
1.2 Get started with Arm® Fortran Compiler	1-15
1.3 Get support	1-17

Chapter 2

Compile and link

2.1 Using the compiler	2-19
2.2 Compile Fortran code for SVE and SVE2-enabled processors	2-22

Chapter 3

Optimize

3.1 Directives	3-24
3.2 Arm Optimization Report	3-31
3.3 Optimization remarks	3-36

Chapter 4

Compiler options

4.1 Arm Fortran Compiler Options by Function	4-41
4.2 ###	4-45
4.3 -armpl=	4-46
4.4 -c	4-48

4.5	-config	4-49
4.6	-cpp	4-50
4.7	-D	4-51
4.8	-E	4-52
4.9	-fassociative-math	4-53
4.10	-fbackslash	4-54
4.11	-fcolor-diagnostics	4-55
4.12	-fconvert=	4-56
4.13	-fcxx-exceptions	4-57
4.14	-fdenormal-fp-math=	4-58
4.15	-fexceptions	4-59
4.16	-ffast-math	4-60
4.17	-ffinite-math-only	4-61
4.18	-ffixed-form	4-62
4.19	-ffixed-line-length-	4-63
4.20	-ffp-contract=	4-64
4.21	-ffree-form	4-65
4.22	-fhonor-infinities	4-66
4.23	-fhonor-nans	4-67
4.24	-finline	4-68
4.25	-finline-functions	4-69
4.26	-finline-hint-functions	4-70
4.27	-fiterative-reciprocal	4-71
4.28	-flto	4-72
4.29	-fmath-errno	4-73
4.30	-fnative-atomics	4-74
4.31	-fno-crash-diagnostics	4-75
4.32	-fno-fortran-main	4-76
4.33	-fopenmp	4-77
4.34	-fopenmp-simd	4-78
4.35	-frealloc-lhs	4-79
4.36	-freciprocal-math	4-80
4.37	-frecursive	4-81
4.38	-fsave-optimization-record	4-82
4.39	-fsign-zero	4-83
4.40	-fsigned-char	4-84
4.41	-fsigned-zeros	4-85
4.42	-fsimdmath	4-86
4.43	-fstack-arrays	4-87
4.44	-fstrict-aliasing	4-88
4.45	-fsyntax-only	4-89
4.46	-ftrapping-math	4-90
4.47	-funsafe-math-optimizations	4-91
4.48	-fvectorize	4-92
4.49	-g	4-93
4.50	-g0	4-94
4.51	-gcc-toolchain=	4-95
4.52	-gline-tables-only	4-96
4.53	-help	4-97
4.54	-help-hidden	4-98

4.55	-I	4-99
4.56	-i8	4-100
4.57	-include	4-101
4.58	-iquote	4-102
4.59	-isysroot	4-103
4.60	-isystem	4-104
4.61	-isystem-after	4-105
4.62	-I	4-106
4.63	-Mnomain	4-107
4.64	-Mstandard	4-108
4.65	-march=	4-109
4.66	-mcpu=	4-110
4.67	-module	4-111
4.68	-no-flang-libs	4-112
4.69	-nocpp	4-113
4.70	-O	4-114
4.71	-o	4-115
4.72	-print-search-dirs	4-116
4.73	-Qunused-arguments	4-117
4.74	-r8	4-118
4.75	-S	4-119
4.76	-shared	4-120
4.77	-static	4-121
4.78	-static-arm-libs	4-122
4.79	-std=	4-123
4.80	-U	4-124
4.81	-v	4-125
4.82	-version	4-126
4.83	-W	4-127
4.84	-Wall	4-128
4.85	-Warm-extensions	4-129
4.86	-Wdeprecated	4-130
4.87	-Wl,	4-131
4.88	-Wp,	4-132
4.89	-w	4-133
4.90	-working-directory	4-134
4.91	-Xassembler	4-135
4.92	-Xlinker	4-136
4.93	-Xpreprocessor	4-137

Chapter 5

Fortran language reference

5.1	Data types and file extensions	5-139
5.2	Intrinsics	5-144
5.3	Statements	5-162

Chapter 6

Standards support

6.1	Fortran 2003	6-170
6.2	Fortran 2008	6-173
6.3	OpenMP 4.0	6-176
6.4	OpenMP 4.5	6-177

Chapter 7

Troubleshoot

7.1	Application segfaults at -Ofast optimization level	7-179
7.2	Compiling with the -fpic option fails when using GCC compilers	7-180
7.3	Error messages when installing Arm® Compiler for Linux	7-181

List of Tables

Arm® Fortran Compiler Developer and reference guide

Table 5-1	<i>Intrinsic data types</i>	5-139
Table 5-2	<i>Supported file extensions</i>	5-140
Table 5-3	<i>Bit manipulation functions and subroutines</i>	5-144
Table 5-4	<i>Elemental character and logical functions</i>	5-145
Table 5-5	<i>Vector and matrix functions</i>	5-147
Table 5-6	<i>Array reduction functions</i>	5-147
Table 5-7	<i>String construction functions</i>	5-149
Table 5-8	<i>Array construction and manipulation functions</i>	5-149
Table 5-9	<i>General inquiry functions</i>	5-150
Table 5-10	<i>Numeric inquiry functions</i>	5-150
Table 5-11	<i>Array inquiry functions</i>	5-151
Table 5-12	<i>Transfer functions</i>	5-151
Table 5-13	<i>Arithmetic functions</i>	5-152
Table 5-14	<i>Miscellaneous functions</i>	5-155
Table 5-15	<i>Subroutines</i>	5-155
Table 5-16	<i>Fortran 2003 functions</i>	5-156
Table 5-17	<i>Fortran 2008 functions</i>	5-157
Table 5-18	<i>Unsupported functions</i>	5-159
Table 5-19	<i>Unsupported subroutines</i>	5-160
Table 5-20	<i>Supported Fortran statements</i>	5-162
Table 6-1	<i>Fortran 2003 support</i>	6-170
Table 6-2	<i>Fortran 2008 support</i>	6-173

<i>Table 6-3</i>	<i>OpenMP 4.0 support</i>	<i>6-176</i>
<i>Table 6-4</i>	<i>OpenMP 4.5 support</i>	<i>6-177</i>

Preface

This preface introduces the *Arm® Fortran Compiler Developer and reference guide*.

It contains the following:

- [About this book on page 11](#).

About this book

Provides information to help you use the Arm Fortran Compiler component of Arm Compiler for Linux. Arm Fortran Compiler is an auto-vectorizing, Linux user-space Fortran compiler, tailored for Server and High Performance Computing (HPC) workloads. Arm Fortran Compiler supports popular Fortran and OpenMP standards and is tuned for Armv8-A based processors.

Using this book

This book is organized into the following chapters:

Chapter 1 Get started

This chapter introduces Arm Fortran Compiler (part of Arm Compiler for Linux and Arm Allinea Studio), and describes how to get started with the compiler, and where to find further support.

Chapter 2 Compile and link

This chapter describes the basic functionality of Arm Fortran Compiler, and describes how to compile your Fortran source with `armflang`.

Chapter 3 Optimize

This chapter describes the optimization-specific features supported in Arm Fortran Compiler.

Chapter 4 Compiler options

This chapter summarizes the supported options used with `armflang`.

Chapter 5 Fortran language reference

Arm Fortran Compiler supports Fortran 2008, Fortran 2003, Fortran 95, Fortran 90, and some Fortran 2018 language features. This chapter can be used as a reference for the supported language features.

Chapter 6 Standards support

Describes the level of support in Arm Fortran Compiler for Fortran and OpenMP standards.

Chapter 7 Troubleshoot

Describes how to diagnose problems when compiling applications using Arm Fortran Compiler.

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the [Arm® Glossary](#) for more information.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

monospace bold

Denotes language keywords when used outside example code.

<and>

Encloses replaceable terms for assembler syntax where they appear in code or code fragments.
For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:

- The title *Arm Fortran Compiler Developer and reference guide*.
- The number 101380_2020_00_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

———— **Note** ————

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- *Arm® Developer*.
- *Arm® Information Center*.
- *Arm® Technical Support Knowledge Articles*.
- *Technical Support*.
- *Arm® Glossary*.

Chapter 1

Get started

This chapter introduces Arm Fortran Compiler (part of Arm Compiler for Linux and Arm Allinea Studio), and describes how to get started with the compiler, and where to find further support.

It contains the following sections:

- [1.1 Arm® Fortran Compiler on page 1-14.](#)
- [1.2 Get started with Arm® Fortran Compiler on page 1-15.](#)
- [1.3 Get support on page 1-17.](#)

1.1 Arm® Fortran Compiler

Arm Fortran Compiler is a Linux user space Fortran compiler for server and High Performance Computing (HPC) Arm-based platforms. It is built on the open-source Flang front-end and the LLVM-based optimization and code generation back-end.

Arm Fortran Compiler supports popular Fortran and OpenMP standards, has a built-in autovectorizer, and is tuned for the 64-bit Armv8-A architecture. It also supports compiling for Scalable Vector Extension (SVE) and SVE2-enabled target platforms.

Arm Fortran Compiler is packaged with Arm C/C++ Compiler and Arm Performance Libraries in a single package called Arm Compiler for Linux. Arm Compiler for Linux is available as part of Arm Allinea Studio.

Arm Allinea Studio is an end-to-end commercial suite for compiling, debugging, and optimizing Linux applications on Arm, and is comprised of Arm Compiler for Linux and Arm Forge.

The Arm Allinea Studio tools require a valid license to use them.

Arm® Fortran Compiler resources

- [Arm Fortran Compiler](#)
- [Arm Allinea Studio](#)
- [Arm Allinea Studio installation instructions](#)
- [Arm Allinea Studio Release history](#)
- [Arm Allinea Studio supported platforms](#)
- [Porting and tuning resources](#)
- [Scalable Vector Extension \(SVE\) information](#)

You can find further help and resources on the [Arm Developer website](#).

Related information

[Arm Fortran Compiler](#)

[Arm Allinea Studio](#)

[Arm Allinea Studio Licensing](#)

1.2 Get started with Arm® Fortran Compiler

Describes how to download and install Arm Compiler for Linux, and how to use Arm Fortran Compiler to compile Fortran source into an executable binary.

Prerequisites

Download and install Arm Compiler for Linux. You can download Arm Compiler for Linux from the [Arm Allinea Studio Downloads](#) page. Learn how to install and configure Arm Compiler for Linux, using the [Arm Compiler for Linux installation instructions](#) on the Arm Developer website.

Procedure

1. Load the environment module for Arm Compiler for Linux:

- a. As part of the installation, Arm recommends that your system administrator makes the Arm Compiler for Linux environment modules available to all users of the tool suite.

To see which environment modules are available on your system, run:

```
module avail
```

Note

If you cannot see the Arm Compiler for Linux environment module, and you know the installation location of the tools, set the MODULEPATH environment variable to include the installation directory:

```
export MODULEPATH=$MODULEPATH:<path/to/installation>/modulefiles/
```

replacing <path/to/installation> with the path to your installation of Arm Compiler for Linux. The default installation location is /opt/arm/.

- b. To load the module for Arm Compiler for Linux, run:

```
module load <architecture>/<linux_variant>/<linux_version>/arm-linux-compiler/  
<version>
```

For example:

```
module load Generic-AArch64/SUSE/12/arm-linux-compiler/20.2
```

- c. Check your environment. Examine the PATH variable. PATH must contain the appropriate bin directory from <path/to/installation>:

```
echo $PATH  
/opt/arm/arm-linux-compiler-20.2_Generic-AArch64_SUSE-  
12_aarch64-linux/bin:...
```

Note

To automatically load the Arm Compiler for Linux every time you log into your Linux terminal, add the module load command for your system and product version to your .profile file.

2. To generate an executable binary, compile your program with Arm Fortran Compiler.

Specify (-o) the output binary file, <binary>, and the input source filename, <source>.<fortran-extension>:

```
armflang -o <binary> <source>.<fortran-extension>
```

Arm Fortran Compiler builds your binary <binary>.

To run your binary, use:

```
./<binary>
```

Example 1-1 Example: Compile and run a Hello World program

This example describes how to write, compile, and run a simple “Hello World” Fortran program.

1. Load the environment module for your system:

```
module load <architecture>/<linux_variant>/<linux_version>/arm-linux-compiler/<version>
```

2. Create a “Hello World” program and save it in an .f90 file, for example: hello.f90:

```
program hello
  print *, 'hello world'
end
```

3. To generate an executable binary, compile your Hello World program with Arm Fortran Compiler.

Specify (-o) the input file, hello.f90, and the binary name, hello:

```
armflang -o <binary-filename> <source-filename>.<fortran-extension>
```

4. Run the generated binary hello:

```
./hello
```

Next Steps

For more information about compiling and linking as separate steps, and how optimization levels effect auto-vectorization, see [Compile and link on page 2-18](#).

1.3 Get support

To see a list of all the supported compiler options in your terminal, use:

```
armflang --help
```

or

```
man armflang
```

A description of each supported command-line option is available in [Compiler options on page 4-38](#).

If you encounter a problem when developing your application and compiling with the Arm Compiler for Linux, see the [Troubleshoot on page 7-178](#) topic.

If you encounter a problem when using Arm Compiler for Linux, contact the Arm Support team:

[Contact Arm Support](#)

Chapter 2

Compile and link

This chapter describes the basic functionality of Arm Fortran Compiler, and describes how to compile your Fortran source with `armflang`.

It contains the following sections:

- [2.1 Using the compiler on page 2-19.](#)
- [2.2 Compile Fortran code for SVE and SVE2-enabled processors on page 2-22.](#)

2.1 Using the compiler

Describes how to compile and link object files and enable optimization options.

Compile and link

To generate an executable binary, compile a program using the `armflang` command. For example, to compile `example1.f90`, and create the binary `example1`, use:

```
armflang -o example1 example1.f90
```

You can also specify multiple source files on a single line. For example, to compile `example1a.f90` and `example1b.f90`, and create the binary `example1.f90`, use:

```
armflang -o example1 example1a.f90 example1b.f90
```

Each source file is compiled individually and then linked into a single executable binary.

To compile each of your source files individually into an object file, specify the `-c` (compile-only) option. For example, to compile `example1a.f90` into `example1a.o`, and to compile `example1b.f90` into `example1b.o`, use:

```
armflang -c -o example1a.o example1a.f90  
armflang -c -o example1b.o example1b.f90
```

To link two object files into an executable binary, run `armflang` with the `-o` option, state the binary name, and pass the object files. For example, to create the binary `example1` from the object files `example1a.o` and `example1b.o`, use:

```
armflang -o example1 example1a.o example1b.o
```

Control compiler optimization

To control the optimization level, use the `-O<level>` option. The `-O0` option is the lowest optimization level. `-O3` is the highest. Arm Fortran Compiler only performs auto-vectorization at `-O2` and higher, and uses `-O0` as the default setting. The optimization flag can be specified when generating a binary, for example:

```
armflang -O3 -o <binary> <source-file>
```

The optimization flag can also be specified when generating an object file. For example, to compile `example1a.f90` into `example1a.o`, and to compile `example1b.f90` into `example1b.o`, both at `-O3` level, use:

```
armflang -O3 -c -o example1a.o example1a.f90  
armflang -O3 -c -o example1b.o example1b.f90
```

It can also be specified when linking object files. For example, to create the binary `example1` from the object files `example1a.o` and `example1b.o`, use:

```
armflang -O3 -o example1 example1a.o example1b.o
```

Compile and optimize using CPU auto-detection

The `-mcpu=native` option enables the compiler to automatically detect the architecture and processor type of the CPU you are running the compiler on, and to enable the optimization available for that target.

For example, to compile `example1.f90` into the binary `example1` with CPU auto-detection, use:

```
armflang -O3 -mcpu=native -o example1 example1.f90
```

This option supports a range of Armv8-A based SoCs, including ThunderX2, Neoverse N1, and A64FX.

Note

The optimizations that are performed according to the auto-detected architecture and processor are independent of the optimization level that is denoted by the `-O<level>` option.

Common compiler options

Describes some common compiler options.

-S

Outputs assembly code, rather than object code. Produces a text `.s` file containing annotated assembly code.

-c

Performs the compilation step, but does not perform the link step. Produces an ELF object `.o` file. To later link the object files into an executable binary, re-run `armflang` and pass in the object files.

-o file

Specifies the name of the output file.

-march=name[+[no]feature]

Targets an architecture profile, generating generic code that runs on any processor of that architecture. For example `-march=armv8-a`, `-march=armv8-a+sve`, or `-march=armv8-a+sve2`.

Note

If you know your target microarchitecture, Arm recommends using the `-mcpu` option instead of `-march`.

-mcpu=native

Enables the compiler to automatically detect the CPU you are running the compiler on, and optimize accordingly. The compiler selects a suitable architecture profile for that CPU. If you use `-mcpu`, you do not need to use the `-march` option.

`mcpu` supports a range of Armv8-A-based System-on-Chips (SoCs), including ThunderX2, Neoverse N1, and A64FX.

Note

When `-mcpu` is not specified, it defaults to `mcpu=generic` which generates portable output suitable for any Armv8-A-based computer.

-Olevel

Specifies the level of optimization to use when compiling source files. The default is `-O0`.

--config /path/to/<config-file>.cfg

Passes the location of a configuration file to the compile command. Use a configuration file to specify a set of compile options to be run at compile time. The configuration file can be passed at compile time, or an environment variable can be set for it to be used for every invocation of the compiler. For more information about creating and using a configuration file, see [Configure Arm Compiler for Linux](#).

--help

Describes the most common options that are supported by Arm Fortran Compiler.

--version

Displays version information.

For a detailed description of all the supported compiler options, see [Compiler options on page 4-38](#).

To view the supported options on the command-line, use the man pages:

```
man armflang
```

Related concepts

[2.2 Compile Fortran code for SVE and SVE2-enabled processors on page 2-22](#)

Related references

[Chapter 4 Compiler options on page 4-38](#)

2.2 Compile Fortran code for SVE and SVE2-enabled processors

You can compile for Scalable Vector Extension (SVE) and Scalable Vector Extension version two (SVE2)-enabled target architectures with Arm Fortran Compiler.

With Arm Compiler for Linux, you can:

- Assemble source code containing SVE and SVE2 instructions.
- Disassemble ELF object files containing SVE and SVE2 instructions.
- Compile C and C++ code for SVE and SVE2-enabled targets, with an advanced auto-vectorizer capable of taking advantage of the SVE and SVE2 features.

To optimize Fortran code for an SVE or SVE2-enabled target, enable auto-vectorization by using optimization level `-O2` or `-O3`, and specify an SVE or SVE2-enabled target architecture using the `-march=` option:

For SVE targets, use the ```-march=armv8-a+sve``` option. For example:

```
armflang -O3 -march=armv8-a+sve -o <binary> <source-file>
```

For SVE2 targets, use the ```-march=armv8-a+sve2``` option. For example:

```
armflang -O3 -march=armv8-a+sve2 -o <binary> <source-file>
```

Note

- `sve2` also enables `sve`.
- There are several SVE2 Cryptographic Extensions available that also enable SVE2: `sve2-aes`, `sve2-bitperm`, `sve2-sha3`, and `sve2-sm4`.
- When enabling either the `sve2` or `sve` features, to link to the SVE-enabled version of Arm Performance Libraries, you must also include the `-armpl=sve` option. For example:

```
armflang -O3 -march=armv8-a+sve -armpl=sve -o <binary> <source-file>
```

For more information about the supported options for `-armpl`, see the `-armpl` description in [Arm Fortran Compiler Options by Function](#) on page 4-41.

- For a full list of supported `-march` options, see [Arm Fortran Compiler Options by Function](#) on page 4-41.

You can also specify multiple source files on a single line. Each source file is compiled individually and then linked into a single executable binary:

For example, to compile for an SVE-enabled target, use:

```
armflang -O3 -march=armv8-a+sve -o <binary> <source-file-1> <source-file-2>
```

For example, to compile for an SVE2-enabled target, use:

```
armflang -O3 -march=armv8-a+sve2 -o <binary> <source-file-1> <source-file-2>
```

To run SVE or SVE2 code on non-SVE platforms, download and install [Arm Instruction Emulator](#). Arm Instruction Emulator runs on AArch64 platforms and emulates SVE and SVE2 instructions, enabling you to prepare your code before running on SVE or SVE2-enabled hardware.

For more information about Arm Instruction Emulator, see the [Arm Instruction Emulator](#) web page.

Related information

[Porting and Optimizing HPC Applications for Arm SVE](#)

Chapter 3

Optimize

This chapter describes the optimization-specific features supported in Arm Fortran Compiler.

It contains the following sections:

- [3.1 Directives on page 3-24.](#)
- [3.2 Arm Optimization Report on page 3-31.](#)
- [3.3 Optimization remarks on page 3-36.](#)

3.1 Directives

Directives are used to provide additional information to the compiler, and to control the compilation of specific code blocks, for example, loops. This chapter describes what directives are supported in Arm Fortran Compiler.

To specify a compiler directive in your source file, use:

- For free-form Fortran, use `!dir$` to indicate a directive, or `!$omp` to indicate an OpenMP directive.
- For fixed-form Fortran, either `!dir$` or `cdir$` can be used to indicate a directive, and either `!$omp` or `c$omp` can be used to indicate an OpenMP directive.

Warning

Directives using `cdir$` or `c$omp` must start from the first column.

Note

To enable OpenMP directives, you must also include the `-fopenmp` compiler option in the compile command line.

For more information about which OpenMP directives are supported, see [Standards support on page 6-169](#). For more information on the `-fopenmp` compiler options, see [Arm Fortran Compiler Options by Function on page 4-41](#).

This section contains the following subsections:

- [3.1.1 ivdep on page 3-24](#).
- [3.1.2 vector always on page 3-25](#).
- [3.1.3 novector on page 3-26](#).
- [3.1.4 omp simd on page 3-27](#).
- [3.1.5 unroll on page 3-28](#).
- [3.1.6 nounroll on page 3-29](#).

3.1.1 ivdep

Apply this general-purpose directive to a loop to force the vectorizer to ignore memory dependencies of iterative loops, and proceed with the vectorization.

Syntax

Command-line option:

None

Source:

```
!dir$ ivdep  
<loops>
```

Note

If you are using fixed-form Fortran, directives can be indicated using `cdir$` or `!dir$`, but must start from the first column.

Parameters

None

Example: Using ivdep

Example usage of the `ivdep` directive.

```
subroutine sum(myarr1,myarr2,ub)
  integer, pointer :: myarr1(:)
  integer, pointer :: myarr2(:)
  integer :: ub
  !dir$ ivdep
  do i=1,ub
    myarr1(i) = myarr1(i)+myarr2(i)
  end do
end subroutine
```

Note

The example uses the free-form syntax. For fixed-form formats, replace `!dir$` with `cdir$`.

Command-line invocation

```
armflang -O3 <test>.f90 -S -Rpass-missed=loop-vectorize
-Rpass=loop-vectorize
```

Outputs

1. With the pragma, the loop that is given below says the following:

```
remark vectorized loop (vectorization width: 2, interleaved
count: 1) [-Rpass=loop-vectorize]
```

2. Without the pragma, the loop that is given below says the following:

```
remark: loop not vectorized [-Rpass-missed=loop-vectorize]
```

3.1.2 vector always

Apply this directive to force vectorization of a loop. The directive tells the vectorizer to ignore any potential cost-based implications.

Note

The loop needs to be able to be vectorized.

Syntax

Command-line option:

None

Source:

```
!dir$ vector always
<loops>
```

Note

If you are using fixed-form Fortran, directives can be indicated using `cdir$` or `!dir$`, but must start from the first column.

Parameters

None

Example: Using vector always

Example usage of the `vector always` directive.

Code example:

```
subroutine add(a,b,c,d,e,ub)
  implicit none
  integer :: i, ub
  integer, dimension(:) :: a, b, c, d, e
  !dir$ vector always
  do i=1, ub
    e(i) = a(c(i)) + b(d(i))
  end do
end subroutine add
```

Note

The example uses the free-form syntax. For fixed-form formats, replace `!dir$` with `cdir$`.

Command-line invocation

```
armflang -O3 <test>.f90 -S -Rpass-missed=loop-vectorize -Rpass=loop-vectorize
```

Outputs

- With the pragma, the output for the example is:

```
remark: vectorized loop (vectorization width: 4, interleaved
count: 1) [-Rpass=loop-vectorize]
```

- Without the pragma, the output for the example is:

```
remark: the cost-model indicates that vectorization is not beneficial [-Rpass-missed=loop-
vectorize]
```

Related references

[3.3 Optimization remarks on page 3-36](#)

3.1.3 novector

Apply this directive to disable vectorization of a loop.

Note

Use this directive when vectorization would cause a regression instead of an improvement.

Syntax

Command-line option:

None

Source:

```
!dir$ novector
<loops>
```

Note

If you are using fixed-form Fortran, directives can be indicated using `cdir$` or `!dir$`, but must start from the first column.

Parameters

None

Example: Using novector

Example usage of the `novector` directive.

Code example:

```
subroutine add(arr1,arr2,arr3,ub)
  integer :: arr1(ub), arr2(ub), arr3(ub)
  integer :: i
  !dir$ novector
  do i=1,ub
    arr1(i) = arr1(i) + arr2(i)
  end do
end subroutine add
```

Note

The example uses the free-form syntax. For fixed-form formats, replace `!dir$` with `cdir$`.

Command-line invocation

```
armflang -O3 <test>.f90 -S -Rpass-missed=loop-vectorize -Rpass=loop-vectorize
```

Outputs

- With the pragma, the output for the example is:

```
remark: loop not vectorized [-Rpass-missed=loop-vectorize]
```

- Without the pragma, the output for the example is:

```
remark: vectorized loop (vectorization width: 4, interleaved count: 2)
[-Rpass=loop-vectorize]
```

Related references

[3.3 Optimization remarks on page 3-36](#)

3.1.4 omp simd

Apply this OpenMP directive to a loop to indicate that the loop can be transformed into a SIMD loop.

Syntax

Command-line option:

`-fopenmp`

Source:

```
!$omp simd
<do-loops>
```

Note

If you are using fixed-form Fortran, OpenMP directives can be indicated using `!$omp` or `c$omp`, but must start from the first column.

Parameters

None

Example: Using omp simd

Example usage of the `omp simd` directive.

Code example:

```
subroutine sum(myarr1,myarr2,myarr3,myarr4,myarr5,ub)
  integer, pointer :: myarr1(:)
  integer, pointer :: myarr2(:)
  integer, pointer :: myarr3(:)
  integer, pointer :: myarr4(:)
  integer, pointer :: myarr5(:)
```

```
integer :: ub
!$omp simd
do i=1,ub
  myarr1(i) = myarr2(myarr4(i))+myarr3(myarr5(i))
end do
end subroutine
```

Command-line invocation

```
armflang -O3 -fopenmp <test>.f90 -S -Rpass-missed=loop-vectorize
-Rpass=loop-vectorize
```

Outputs

1. With the pragma, the loop that is given below says the following:

```
remark vectorized loop (vectorization width: 2, interleaved
count: 1) [-Rpass=loop-vectorize]
```

2. Without the pragma, the loop that is given below says the following:

```
remark: loop not vectorized [-Rpass-missed=loop-vectorize]
```

Related references

[Chapter 6 Standards support on page 6-169](#)

3.1.5 unroll

Instructs the compiler optimizer to unroll a DO loop when optimization is enabled with the compiler optimization flags `-O2` or higher.

Syntax

Command-line option:

None

Source:

```
!dir$ unroll
<loops>
```

Note

If you are using fixed-form Fortran, directives can be indicated using `cdir$` or `!dir$`, but must start from the first column.

Parameters

None

Example: Using unroll

Example usage of the `unroll` directive.

Code example:

```
subroutine add(a,b,c,d)
integer, parameter :: m = 1000
integer :: a(m), b(m), c(m), d(m)
integer :: i
!DIR$ UNROLL
do i =1, m
  b(i) = a(i) + 1
  d(i) = c(i) + 1
end do
end subroutine add
```

Note

The example uses the free-form syntax. For fixed-form formats, replace `!dir$` with `cdir$`.

Related references

[3.1.6 nounroll on page 3-29](#)

[3.3 Optimization remarks on page 3-36](#)

[4.1 Arm Fortran Compiler Options by Function on page 4-41](#)

3.1.6 nounroll

Prevents the unrolling of DO loops when optimization is enabled with the compiler optimization flags `-O2` or higher.

Syntax

Command-line option:

None

Source:

```
!dir$ nounroll
<loops>
```

Note

If you are using fixed-form Fortran, directives can be indicated using `cdir$` or `!dir$`, but must start from the first column..

Parameters

None

Example: Using nounroll

Example usage of the `nounroll` directive.

Code example:

```
subroutine add(a,b,c,d)
  integer, parameter :: m = 1000
  integer :: a(m), b(m), c(m), d(m)
  integer :: i
  !DIR$ NOUNROLL
  do i =1, m
    b(i) = a(i) + 1
    d(i) = c(i) + 1
  end do
end subroutine add
```

Note

The example uses the free-form syntax. For fixed-form formats, replace `!dir$` with `cdir$`.

Related references

[3.1.5 unroll on page 3-28](#)

[3.3 Optimization remarks on page 3-36](#)

[4.1 Arm Fortran Compiler Options by Function on page 4-41](#)

Related references

[3.1.1 ivdep on page 3-24](#)

[3.1.2 vector always on page 3-25](#)

[3.1.3 novector on page 3-26](#)

[3.1.4 omp simd on page 3-27](#)

[3.1.5 unroll on page 3-28](#)

[3.1.6 nounroll on page 3-29](#)

3.2 Arm Optimization Report

Arm Optimization Report builds on the `llvm-opt-report` tool available in open-source LLVM. Arm Optimization Report shows you the optimization decisions that the compiler is making, in-line with your source code, enabling you to better understand the unrolling, vectorization, and interleaving behavior.

Unrolling

Example questions: Was a loop unrolled? If so, what was the unroll factor?

Unrolling is when a scalar loop is transformed to perform multiple iterations at once, but still as scalar instructions.

The unroll factor is the number of iterations of the original loop that are performed at once. Sometimes, loops with known small iteration counts are completely unrolled, such that no loop structure remains. In completely unrolled cases, the unroll factor is the total scalar iteration count.

Vectorization

Example questions: Was a loop vectorized? If so, what was the vectorization factor?

Vectorization is when multiple iterations of a scalar loop are replaced by a single iteration of vector instructions.

The vectorization factor is the number of lanes in the vector unit, and corresponds to the number of scalar iterations that are performed by each vector instruction

Note

The true vectorization factor is unknown at compile-time for SVE, because SVE supports scalable vectors.

When SVE is enabled, Arm Optimization Report reports a vectorization factor that corresponds to a 128-bit SVE implementation.

If you are working with an SVE implementation with a larger vector width (for example, 256 bits or 512 bits), the number of scalar iterations that are performed by each vector instruction increases proportionally.

`SVE scaling factor = <true SVE vector width> / 128`

Loops vectorized using scalable vectors are annotated with `VS<F,I>`. For more information, see [arm-opt-report reference on page 3-33](#).

Interleaving

Example question: What was the interleave count?

Interleaving is a combination of vectorization followed by unrolling; multiple streams of vector instructions are performed in each iteration of the loop.

The combination of vectorization and unrolling information tells you how many iterations of the original scalar loop are performed in each iteration of the generated code.

```
Number of scalar iterations = <unroll factor> x <vectorization factor> x <interleave count>
x <SVE scaling factor>
```

Reference

The annotations Arm Optimization Report uses to annotate the source code, and the options that can be passed to `arm-opt-report` are described in the **Arm Optimization Report reference**.

This section contains the following subsections:

- [3.2.1 How to use Arm Optimization Report on page 3-32](#).
- [3.2.2 arm-opt-report reference on page 3-33](#).

3.2.1 How to use Arm Optimization Report

This topic describes how to use Arm Optimization Report.

Prerequisites

Download and install Arm Compiler for Linux version 20.0+. For more information, see [Download Arm Compiler for Linux](#) and [Installation](#).

Procedure

1. To generate a machine-readable .opt.yaml report, at compile time add -fsave-optimization-record to your command line.

The <filename>.opt.yaml report is generated by Arm Compiler, where <filename> is the name of the binary.

2. To inspect the <filename>.opt.yaml report, as augmented source code, use arm-opt-report:

```
arm-opt-report <filename>.opt.yaml
```

Annotated source code appears in the terminal.

Example 3-1 Example

1. Create an example file called example.f90 containing the following code:

```
subroutine foo
  implicit none
  call bar()
end subroutine foo
subroutine test
  implicit none
  integer :: i
  integer, dimension(1600) :: res, p, d
  do i = 1, 1600
    res(i) = merge(res(i), res(i) + d(i), p(i) == 0)
  end do
  do i = 1, 16
    res(i) = merge(res(i), res(i) + d(i), p(i) == 0)
  end do
  call foo()
  call foo()
  call bar()
  call foo()
end subroutine test
```

2. Compile the file, for example to a shared object called example.o:

```
armflang -O3 -fsave-optimization-record -c -o example.o example.f90
```

This generates a file, example.opt.yaml, in the same directory as the built object.

For compilations that create multiple object files, there is a report for each build object.

————— Note —————

This example compiles to a shared object, however, you could also compile to a static object or to a binary.

3. View the example.opt.yaml file using arm-opt-report:

```
arm-opt-report example.opt.yaml
```

Annotated source code is displayed in the terminal:

```
< example.f90
1      subroutine foo
2          implicit none
3          call bar()
4      end subroutine foo
5
6      subroutine test
```



```

 7      implicit none
 8      integer :: i
 9      integer, dimension(1600) :: res, p, d
10
11
12      do i = 1, 1600
13          res(i) = merge(res(i), res(i) + d(i), p(i) == 0)
14      end do
15
16
17      do i = 1, 16
18          res(i) = merge(res(i), res(i) + d(i), p(i) == 0)
19      end do
20
21 I      call foo()
22 I      call foo()
23      call bar()
24 I      call foo()
25  end subroutine test

```

The example Arm Optimization Report output is interpreted as follows:

- The do loop on line 12:
 - Is vectorized
 - Has a vectorization factor of four (there are four 32-bit integer lanes)
 - Has an interleave factor of two (the loop was unrolled twice)
- The for loop on line 19 is unrolled 16 times. This means it is completely unrolled, with no remaining loops.
- All three instances of `call foo()` are inlined

Related references

[3.2.2 arm-opt-report reference on page 3-33](#)

Related information

[Arm Compiler for Linux and Arm Allinea Studio](#)

[Take a trial](#)

[Help and tutorials](#)

3.2.2 arm-opt-report reference

Arm Optimization Report (`arm-opt-report`) is a tool to generate an optimization report from YAML optimization record files.

`arm-opt-report` uses a YAML optimization record, as produced by the `-fsave-optimization-record` option of LLVM, to output annotated source code that shows the various optimization decisions taken by the compiler.

Note

`-fsave-optimization-record` is not set by default by Arm Compiler for Linux.

Possible annotations are:

Annotation	Description
I	A function was inlined.
U<N>	A loop was unrolled <N> times.

(continued)

Annotation	Description
V<F, I>	<p>A loop has been vectorized.</p> <p>Each vector iteration performed has the equivalent of $F \times I$ scalar iterations.</p> <p>Vectorization Factor, F, is the number of scalar elements that are processed in parallel.</p> <p>Interleave count, I, is the number of times the vector loop was unrolled.</p>
VS<F, I>	<p>A loop has been vectorized using scalable vectors.</p> <p>Each vector iteration performed has the equivalent of $N \times F \times I$ scalar iterations, where N is the number of vector granules, which can vary according to the machine the program is run on.</p> <p>For example, LLVM assumes a granule size of 128 bits when targeting SVE.</p> <p>F (Vectorization Factor) and I (Interleave count) are as described for V<F, I>.</p>

Syntax

```
arm-opt-report [options] <input>
```

Options**Generic Options:****--help**

Displays the available options (use --help-hidden for more).

--help-list

Displays a list of available options (--help-list-hidden for more).

--version

Displays the version of this program.

llvm-opt-report options:**--hide-detrimental-vectorization-info**

Hides remarks about vectorization being forced despite the cost-model indicating that it is not beneficial.

--hide-inline-hints

Hides suggestions to inline function calls which are preventing vectorization.

--hide-lib-call-remark

Hides remarks about the calls to library functions that are preventing vectorization.

--hide-vectorization-cost-info

Hides remarks about the cost of loops that are not beneficial for vectorization.

--no-demangle

Does not demangle function names.

-o=<string>

Specifies an output file to write the report to.

-r=<string>

Specifies the root for relative input paths.

-s

Omits vectorization factors and associated information.

--strip-comments

Removes comments for brevity

--strip-comments=<arg>

Removes comments for brevity. Arguments are:

- none: Do not strip comments.
- c: Strip C-style comments.
- c++: Strip C++-style comments.
- fortran: Strip Fortran-style comments.

Outputs

Annotated source code.

Related tasks

[3.2.1 How to use Arm Optimization Report on page 3-32](#)

Related tasks

[3.2.1 How to use Arm Optimization Report on page 3-32](#)

Related references

[3.2.2 arm-opt-report reference on page 3-33](#)

3.3 Optimization remarks

Optimization remarks provide you with information about the choices made by the compiler. You can use them to see which code has been inlined or they can help you understand why a loop has not been vectorized.

By default, Arm Fortran Compiler prints compilation information to `stderr`. Optimization remarks print this optimization information to the terminal, or you can choose to pipe them to an output file.

To enable optimization remarks, choose from following `Rpass` options:

- `-Rpass=<regex>`: Information about what the compiler has optimized.
- `-Rpass-analysis=<regex>`: Information about what the compiler has analyzed.
- `-Rpass-missed=<regex>`: Information about what the compiler failed to optimize.

For each option, replace `<regex>` with an expression for the type of remarks you wish to view.

Recommended `<regex>` queries are:

- `-Rpass=(loop-vectorize|inline|loop-unroll)`
- `-Rpass-missed=(loop-vectorize|inline|loop-unroll)`
- `-Rpass-analysis=(loop-vectorize|inline|loop-unroll)`

where `loop-vectorize` filters remarks regarding vectorized loops, `inline` for remarks regarding inlining, and `loop-unroll` for remarks about unrolled loops.

Note

To search for all remarks, use the expression `.*`. Use this expression with caution; depending on the size of code, and the level of optimization, a lot of information can print.

When you provide `-Rpass`, `armflang` generates debug line tables equivalent to passing `-gline-tables-only`, unless you instruct it not to by another debug controlling option. This default behavior ensures that source location information is available to print the remarks.

To compile with optimization remarks enabled, request debug information, and pipe the information to an output file, pass the selected options and debug information to `armflang`, and use `> <output-file>`:

```
armflang -O<level> -Rpass[<-<option>>=<regex> <source-file> [<debug-option>] 2> <output-file>
```

This section contains the following subsection:

- [3.3.1 Enable optimization remarks on page 3-36.](#)

3.3.1 Enable optimization remarks

Describes how to enable optimization remarks and to investigate the choices made by the compiler.

Procedure

1. Compile your code and use the `-Rpass=<regex>`, `-Rpass-missed=<regex>`, or `Rpass-analysis=<regex>` optimization remark options with the `-g` or `gline-tables-only` debug options:

```
armflang -O<level> -Rpass[<-<option>>=<regex> <source-file> [<debug-option>]
```

For example, to enable optimization remarks to be reported for an `example.f90` input file, use:

```
armflang -O3 -Rpass=loop-vectorize example.F90 -gline-tables-only
```

Result:

```
example.F90:21: vectorized loop (vectorization width: 2,
interleaved count: 1)
[-Rpass=loop-vectorize]
do i=1
```

2. Pipe the loop vectorization optimization remarks to a file:

```
armflang -O<level> -Rpass[<option>]=<regex> <source-file> [<debug-option>] 2> <output-file>
```

For example, to pipe to a file called `vecreport.txt`, use:

```
armflang -O3 -Rpass=loop-vectorize -Rpass-analysis=loop-vectorize -Rpass-missed=loop-vectorize example.F90 -gline-tables-only 2> vecreport.txt
```

A `vecreport.txt` file is output with the optimization remarks in it.

Related information

Arm Fortran Compiler

Related tasks

3.3.1 Enable optimization remarks on page 3-36

Chapter 4

Compiler options

This chapter summarizes the supported options used with `armflang`.

`armflang` provides many command-line options, including most Flang command-line options in addition to a number of Arm-specific options. Flang is a Fortran language front-end integrated with LLVM which supports the community feature command-line options that are also supported by Clang. These options are described in the LLVM documentation on the LLVM Compiler Infrastructure Project web site, <http://llvm.org>.

It contains the following sections:

- [4.1 Arm Fortran Compiler Options by Function on page 4-41.](#)
- [4.2 -#### on page 4-45.](#)
- [4.3 -armpl= on page 4-46.](#)
- [4.4 -c on page 4-48.](#)
- [4.5 -config on page 4-49.](#)
- [4.6 -cpp on page 4-50.](#)
- [4.7 -D on page 4-51.](#)
- [4.8 -E on page 4-52.](#)
- [4.9 -fassociative-math on page 4-53.](#)
- [4.10 -fbackslash on page 4-54.](#)
- [4.11 -fcolor-diagnostics on page 4-55.](#)
- [4.12 -fconvert= on page 4-56.](#)
- [4.13 -fcxx-exceptions on page 4-57.](#)
- [4.14 -fdenormal-fp-math= on page 4-58.](#)
- [4.15 -fexceptions on page 4-59.](#)
- [4.16 -ffast-math on page 4-60.](#)
- [4.17 -ffinite-math-only on page 4-61.](#)
- [4.18 -ffixed-form on page 4-62.](#)

- 4.19 *-ffixed-line-length-* on page 4-63.
- 4.20 *-ffp-contract=* on page 4-64.
- 4.21 *-ffree-form* on page 4-65.
- 4.22 *-fhonor-infinities* on page 4-66.
- 4.23 *-fhonor-nans* on page 4-67.
- 4.24 *-finline* on page 4-68.
- 4.25 *-finline-functions* on page 4-69.
- 4.26 *-finline-hint-functions* on page 4-70.
- 4.27 *-fiterative-reciprocal* on page 4-71.
- 4.28 *-flto* on page 4-72.
- 4.29 *-fmath-errno* on page 4-73.
- 4.30 *-fnative-atomics* on page 4-74.
- 4.31 *-fno-crash-diagnostics* on page 4-75.
- 4.32 *-fno-fortran-main* on page 4-76.
- 4.33 *-fopenmp* on page 4-77.
- 4.34 *-fopenmp-simd* on page 4-78.
- 4.35 *-frealloc-lhs* on page 4-79.
- 4.36 *-freciprocal-math* on page 4-80.
- 4.37 *-frecursive* on page 4-81.
- 4.38 *-fsave-optimization-record* on page 4-82.
- 4.39 *-fsign-zero* on page 4-83.
- 4.40 *-fsigned-char* on page 4-84.
- 4.41 *-fsigned-zeros* on page 4-85.
- 4.42 *-fsimdmath* on page 4-86.
- 4.43 *-fstack-arrays* on page 4-87.
- 4.44 *-fstrict-aliasing* on page 4-88.
- 4.45 *-fsyntax-only* on page 4-89.
- 4.46 *-ftrapping-math* on page 4-90.
- 4.47 *-funsafe-math-optimizations* on page 4-91.
- 4.48 *-fvectorize* on page 4-92.
- 4.49 *-g* on page 4-93.
- 4.50 *-g0* on page 4-94.
- 4.51 *-gcc-toolchain=* on page 4-95.
- 4.52 *-gline-tables-only* on page 4-96.
- 4.53 *-help* on page 4-97.
- 4.54 *-help-hidden* on page 4-98.
- 4.55 *-I* on page 4-99.
- 4.56 *-i8* on page 4-100.
- 4.57 *-include* on page 4-101.
- 4.58 *-iquote* on page 4-102.
- 4.59 *-isysroot* on page 4-103.
- 4.60 *-isystem* on page 4-104.
- 4.61 *-isystem-after* on page 4-105.
- 4.62 *-l* on page 4-106.
- 4.63 *-Mnomain* on page 4-107.
- 4.64 *-Mstandard* on page 4-108.
- 4.65 *-march=* on page 4-109.
- 4.66 *-mcpu=* on page 4-110.
- 4.67 *-module* on page 4-111.
- 4.68 *-no-flang-libs* on page 4-112.
- 4.69 *-nocpp* on page 4-113.
- 4.70 *-O* on page 4-114.
- 4.71 *-o* on page 4-115.
- 4.72 *-print-search-dirs* on page 4-116.
- 4.73 *-Qunused-arguments* on page 4-117.
- 4.74 *-r8* on page 4-118.

- 4.75 *-S* on page 4-119.
- 4.76 *-shared* on page 4-120.
- 4.77 *-static* on page 4-121.
- 4.78 *-static-arm-libs* on page 4-122.
- 4.79 *-std=* on page 4-123.
- 4.80 *-U* on page 4-124.
- 4.81 *-v* on page 4-125.
- 4.82 *-version* on page 4-126.
- 4.83 *-W* on page 4-127.
- 4.84 *-Wall* on page 4-128.
- 4.85 *-Warm-extensions* on page 4-129.
- 4.86 *-Wdeprecated* on page 4-130.
- 4.87 *-Wl*, on page 4-131.
- 4.88 *-Wp*, on page 4-132.
- 4.89 *-w* on page 4-133.
- 4.90 *-working-directory* on page 4-134.
- 4.91 *-Xassembler* on page 4-135.
- 4.92 *-Xlinker* on page 4-136.
- 4.93 *-Xpreprocessor* on page 4-137.

4.1 Arm Fortran Compiler Options by Function

This provides a summary of the `armflang` command-line options that Arm Fortran Compiler supports.

Actions

Options that control what action to perform on the input.

Option	Description
4.8 -E on page 4-52	Only run the preprocessor.
4.75 -S on page 4-119	Only run preprocess and compilation steps.
4.4 -c on page 4-48	Only run preprocess, compile, and assemble steps.
4.13 -fcxx-exceptions on page 4-57	Enable C++ exceptions.
4.15 -fexceptions on page 4-59	Enable support for exception handling.
4.25 -finline-functions on page 4-69	Inline suitable functions.
4.26 -finline-hint-functions on page 4-70	Inline functions which are (explicitly or implicitly) marked inline.
4.33 -fopenmp on page 4-77	Enable OpenMP and link in the OpenMP library, libomp.
4.34 -fopenmp-simd on page 4-78	Enable processing of ‘ <code>simd</code> ’ and the ‘ <code>declare simd</code> ’ pragma, without enabling OpenMP or linking in the OpenMP library, libomp. Enabled by default.
4.45 -fsyntax-only on page 4-89	Show syntax errors but do not perform any compilation.

File options

Options that specify input or output files.

Option	Description
4.55 -I on page 4-99	Add directory to include search path. Directories specified with the <code>-I</code> option apply to both the quote form of the include directive and the system header form.
4.5 -config on page 4-49	Passes the location of a configuration file to the compile command.
4.57 -include on page 4-101	Include file before parsing.
4.58 -iquote on page 4-102	Add directory to the include search path.
4.59 -isysroot on page 4-103	For header files, set the system root directory (usually <code>/</code>).
4.60 -isystem on page 4-104	Override the system include directory.
4.61 -isystem-after on page 4-105	Add directory to end of the SYSTEM include search path.
4.71 -o on page 4-115	Write output to <code><file></code> .
4.90 -working-directory on page 4-134	Resolve file paths relative to the specified directory.

Basic driver options

Options that affect basic functionality of the `armclang` or `armflang` driver.

Option	Description
4.2 -### on page 4-45	Print (but do not run) the commands to run for this compilation.
4.51 -gcc-toolchain= on page 4-95	Use the gcc toolchain at the given directory.
4.53 -help on page 4-97	Display available options.
4.54 -help-hidden on page 4-98	Display hidden options. Only use these options if advised to do so by your Arm representative.

(continued)

Option	Description
4.72 -print-search-dirs on page 4-116	Print the paths that are used for finding libraries and programs.
4.81 -v on page 4-125	Show commands to run and use verbose output.
4.82 -version on page 4-126	Show the version number and some other basic information about the compiler.

Optimization options

Options that control what optimizations should be performed.

Option	Description
4.70 -O on page 4-114	Specifies the level of optimization to use when compiling source files.
4.3 -armpl= on page 4-46	Enable Arm Performance Libraries (ArmPL) (disabled by default).
4.9 -fassociative-math on page 4-53	Allow (-fassociative-math) or prevent (-fno-associative-math) the re-association of operands in a series of floating-point operations. Default is -fno-associative-math.
4.14 -fdenormal-fp-math= on page 4-58	Specify the denormal numbers the code is allowed to require.
4.16 -ffast-math on page 4-60	Allow aggressive, lossy floating-point optimizations (disabled by default).
4.17 -ffinite-math-only on page 4-61	Enable optimizations that ignore the possibility of NaN and +/-Inf (disabled by default).
4.20 -ffp-contract= on page 4-64	Controls when the compiler is permitted to form fused floating-point operations (for example, FMAs).
4.22 -fhonor-infinities on page 4-66	Do not allow optimizations that assume the arguments and results of floating point arithmetic are not +/-Inf.
4.23 -fhonor-nans on page 4-67	Do not allow optimizations that assume the arguments and results of floating point arithmetic are not NaN.
4.24 -finline on page 4-68	Enable/disable inlining (enabled by default).
4.27 -fiterative-reciprocal on page 4-71	Allow optimizations that replace division by reciprocal estimation and refinement.
4.28 -flto on page 4-72	Enable (-flto) or disable (-fno-lto) Link Time Optimizations (LTO).
4.29 -fmath-errno on page 4-73	Require math functions to indicate errors.
4.36 -freciprocal-math on page 4-80	Allow division operations to be reassociated
4.38 -fsave-optimization-record on page 4-82	Enable (-fsave-optimization-record) or disable (-fno-save-optimization-record) the generation of a YAML optimization record file. Default is -fno-save-optimization-record.
4.40 -fsigned-char on page 4-84	Set the type of 'char' to be signed. Disabled by default.
4.41 -fsigned-zeros on page 4-85	Do not allow optimizations that ignore the sign of floating point zeros. Enabled by default.
4.42 -fsimdmath on page 4-86	Enables the vectorized libm library to support the vectorization of loops containing calls to basic library functions, such as those declared in math.h
4.44 -fstrict-aliasing on page 4-88	Tells the compiler to adhere to the aliasing rules defined in the source language (enabled by default when using '-Ofast').
4.46 -ftrapping-math on page 4-90	Tell the compiler to assume (-ftrapping-math), or not to assume (-fno-trapping-math), that floating point operations can trap.
4.47 -funsafe-math-optimizations on page 4-91	Enable a subset of the optimizations allowed by -ffast-math.

(continued)

Option	Description
4.48 -fvectorize on page 4-92	Enable/disable loop vectorization (enabled by default).
4.65 -march= on page 4-109	Specifies the base architecture and extensions available on the target.
4.66 -mcpu= on page 4-110	Select which CPU architecture to optimize for.

C/C++ Options

Options that affect the way C workloads are compiled.

Option	Description
4.79 -std= on page 4-123	Language standard to compile for.

Fortran Options

Options that affect the way Fortran workloads are compiled.

Option	Description
4.63 -Mnomain on page 4-107	Do not link in Fortran main.
4.64 -Mstandard on page 4-108	Check Fortran standard conformance.
4.6 -cpp on page 4-50	Preprocess Fortran files.
4.10 -fbackslash on page 4-54	Treat backslash as C-style escape character (-fbackslash) or as a normal character (-fno-backslash) (C-style is the default).
4.12 -fconvert= on page 4-56	Convert between big and little endian data format. Default is -fconvert=native.
4.18 -ffixed-form on page 4-62	Force fixed-form format Fortran. This is default for .f and .F files, and is the inverse of -ffree-form.
4.19 -ffixed-line-length- on page 4-63	Set line length in fixed-form format Fortran. Default is 72. 0 and none are equivalent and set the line length to a very large value (>132).
4.21 -ffree-form on page 4-65	Force free-form format for Fortran. This is default for .f90 and .F90 files, and is the inverse of -ffixed-form.
4.30 -fnative-atomics on page 4-74	Enable (-fnative-atomics) or disable (-fno-native-atomics) the use of native atomic instructions for OpenMP atomics. Default is -fnative-atomics.
4.32 -fno-fortran-main on page 4-76	Do not link in Fortran main.
4.35 -frealloc-lhs on page 4-79	Select semantics for assignments to allocatables.
4.37 -frecursive on page 4-81	Allocate all local arrays on the stack, allowing thread-safe recursion (enabled by default with -fopenmp).
4.39 -fsign-zero on page 4-83	Write floating point numbers of the value zero with the sign bit as a negative number. Enabled by default.
4.43 -fstack-arrays on page 4-87	Place all automatic arrays on stack memory (enabled by default with -Ofast).
4.56 -i8 on page 4-100	Treat INTEGER and LOGICAL as INTEGER*8 and LOGICAL*8.
4.67 -module on page 4-111	Fortran module path.
4.68 -no-flang-libs on page 4-112	Do not link against Flang libraries.
4.69 -nocpp on page 4-113	Do not preprocess Fortran files.
4.74 -r8 on page 4-118	Treat REAL as REAL*8.

Development options

Options that facilitate code development.

Option	Description
4.11 -fcolor-diagnostics on page 4-55	Use colors in diagnostics.
4.49 -g on page 4-93	Generate source-level debug information.
4.50 -g0 on page 4-94	Disable generation of source-level debug information (default).
4.52 -gline-tables-only on page 4-96	Emit debug line number tables only.

Warning options

Options that control the behavior of warnings.

Option	Description
4.73 -Qunused-arguments on page 4-117	Do not emit a warning for unused driver arguments.
4.83 -W on page 4-127	Enable the specified warning.
4.84 -Wall on page 4-128	Enable all warnings.
4.85 -Warm-extensions on page 4-129	Enable warnings about the use of non-standard language features supported by armclang or armflang
4.86 -Wdeprecated on page 4-130	Enable warnings for deprecated constructs and define __DEPRECATED.
4.31 -fno-crash-diagnostics on page 4-75	Disable the auto-generation of preprocessed source files and a script for reproduction during a clang crash.
4.89 -w on page 4-133	Suppress all warnings.

Preprocessor options

Options controlling the behavior of the preprocessor.

Option	Description
4.7 -D on page 4-51	Define <macro> to <value> (or 1 if <value> omitted).
4.80 -U on page 4-124	Undefine macro <macro>.
4.88 -Wp, on page 4-132	Pass the comma separated arguments in <arg> to the preprocessor
4.93 -Xpreprocessor on page 4-137	Pass <arg> to the preprocessor.

Linker options

Options that are passed on to the linker or affect linking.

Option	Description
4.87 -Wl, on page 4-131	Pass the comma separated arguments in <arg> to the linker.
4.91 -Xassembler on page 4-135	Pass <arg> to the assembler.
4.92 -Xlinker on page 4-136	Pass <arg> to the linker.
4.62 -l on page 4-106	Search for the library named <library> when linking.
4.76 -shared on page 4-120	Create a shared object that can be linked against.
4.77 -static on page 4-121	Link against static libraries.
4.78 -static-arm-libs on page 4-122	Link against static Arm libraries.

4.2 -###

Print (but do not run) the commands to run for this compilation.

Syntax

```
armflang -###
```

4.3 -armpl=

Enable Arm Performance Libraries (ArmPL) (disabled by default).

Instructs the compiler to load the optimum version of Arm Performance Libraries for your target architecture and implementation. This option also enables optimized versions of the C mathematical functions declared in the math.h library, tuned scalar and vector implementations of Fortran math intrinsics. This option implies -fsimdmath.

ArmPL provides libraries suitable for a range of supported CPUs. If you intend to use -armpl, you must also specify the required architecture using the -mcpu flag.

The -armpl option also enables:

- Optimized versions of the C mathematical functions declared in math.h.
- Optimized versions of Fortran math intrinsics.
- Auto-vectorization of C mathematical functions (disable this with -fno-simdmath).
- Auto-vectorization of Fortran math intrinsics (disable this with -fno-simdmath).

Default

By default, -armpl is not set (in other words, OFF)

Default argument behavior

If -armpl is set with no arguments, the default behavior of the option is `armpl=lp64,sequential`.

However, the default behavior of the arguments is also determined by the specification (or not) of the -i8 (when using armflang) and -fopenmp options:

- If the -i8 option is not specified, `lp64` is enabled by default. If -i8 is specified, `ilp64` is enabled by default.
- If the -fopenmp option is not specified, `sequential` is enabled by default. If -fopenmp is specified, `parallel` is enabled by default.

In other words:

- Specifying -armpl sets `armpl=lp64,sequential`.
- Specifying -armpl and -i8 sets `armpl=ilp64,sequential`.
- Specifying -armpl and -fopenmp sets `armpl=lp64,parallel`.
- Specifying -armpl, -i8, and -fopenmp sets `armpl=ilp64,parallel`.

Syntax

```
armflang -armpl=<arg1>,<arg2>...
```

Arguments

lp64

Use 32-bit integers. (default)

ilp64

Use 64-bit integers. Inverse of lp64. (default if using -i8 with armflang).

sequential

Use the single-threaded implementation of Arm Performance Libraries. (default)

parallel

Use the OpenMP multi-threaded implementation of Arm Performance Libraries. Inverse of sequential. (default if using -fopenmp)

sve

Use the ‘Generic’ SVE library from Arm Performance Libraries.

Note:

- To enable SVE compilation and library usage on SVE-enabled targets, use `-armpl -mcpu=native`.
- To enable SVE(2) compilation and library usage on a target without native support for these features, use `-armpl=sve -march=armv8-a+<Feature>`, where <Feature> is one of `sve`, `sve2`, `sve2-bitperm`, `sve2-aes`, `sve2-sha3`, or `sve2-sm4`.

4.4 -c

Only run preprocess, compile, and assemble steps.

Syntax

```
armclang -c
```


4.5 -config

Passes the location of a configuration file to the compile command.

Use a configuration file to specify a set of compile options to be run at compile time. The configuration file can be passed at compile time, or an environment variable can be set for it to be used for every invocation of the compiler. For more information about creating and using a configuration file, see <https://developer.arm.com/tools-and-software/server-and-hpc/arm-architecture-tools/arm-allinea-studio/installation/configure>.

Syntax

```
armflang --config <arg>
```

4.6 -cpp

Preprocess Fortran files.

Syntax

```
armflang -cpp
```

4.7 -D

Define <macro> to <value> (or 1 if <value> omitted).

Syntax

```
armflang -D<macro>=<value>
```

4.8 -E

Only run the preprocessor.

Syntax

```
armclang -E
```

4.9 -fassociative-math

Allow (-fassociative-math) or prevent (-fno-associative-math) the re-association of operands in a series of floating-point operations. Default is -fno-associative-math.

For example, $(a * b) + (a * c) \Rightarrow a * (b + c)$. Note: Using -fassociative-math violates the ISO C and C++ language standard.

Default

Default is -fno-associative-math.

Syntax

```
armflang -fassociative-math, -fno-associative-math
```

4.10 -fbackslash

Treat backslash as C-style escape character (-fbackslash) or as a normal character (-fno-backslash) (C-style is the default).

Syntax

```
armflang -fbackslash, -fno-backslash
```

4.11 -fcolor-diagnostics

Use colors in diagnostics.

Syntax

```
armflang -fcolor-diagnostics, -fno-color-diagnostics
```

4.12 -fconvert=

Convert between big and little endian data format. Default is -fconvert=native.

Default

Default is -fconvert=native.

Syntax

```
armflang -fconvert={native \ | swap \ | big-endian \ | little-endian}
```


4.13 -fcxx-exceptions

Enable C++ exceptions.

Syntax

```
armflang -fcxx-exceptions
```

4.14 -fdenormal-fp-math=

Specify the denormal numbers the code is allowed to require.

Syntax

```
armflang -fdenormal-fp-math=<arg>
```

Arguments

ieee

IEEE 754 denormal numbers.

preserve-sign

Flushed-to-zero number signs are preserved in the sign of 0.

positive-zero

Flush denormal numbers to positive zero.

4.15 -fexceptions

Enable support for exception handling.

Syntax

```
armflang -fexceptions
```

4.16 -ffast-math

Allow aggressive, lossy floating-point optimizations (disabled by default).

Using -ffast-math is equivalent to specifying the following flags individually:

-fassociative-math, -ffinite-math-only, -ffp-contract=fast, -fno-math-errno, -fno-signed-zeros, -fno-trapping-math, and -freciprocal-math

Syntax

```
armflang -ffast-math
```

4.17 -ffinite-math-only

Enable optimizations that ignore the possibility of NaN and +/-Inf (disabled by default).

Syntax

```
armflang -ffinite-math-only
```

4.18 -ffixed-form

Force fixed-form format Fortran. This is default for .f and .F files, and is the inverse of -ffree-form.

Syntax

```
armflang -ffixed-form
```

4.19 -ffixed-line-length-

Set line length in fixed-form format Fortran. Default is 72. 0 and none are equivalent and set the line length to a very large value (>132).

Default

Default is -ffixed-line-length-72.

Syntax

```
armflang -ffixed-line-length-{0 \ | 72 \ | 132 \ | none}
```

4.20 -ffp-contract=

Controls when the compiler is permitted to form fused floating-point operations (for example, FMAs).

These instructions typically operate to a higher degree of accuracy than individual multiply and add instructions.

Syntax

```
armflang -ffp-contract={fast\|on\|off}
```

Arguments

fast

Always (default for Fortran workloads). Note: They are not strictly allowed according to the C/C++ standard because they can lead to deviations from the expected results.

on

Only in the presence of the FP_CONTRACT pragma (default for C/C++ workloads).

off

Never.

4.21 -ffree-form

Force free-form format for Fortran. This is default for .f90 and .F90 files, and is the inverse of -ffixed-form.

Syntax

```
armflang -ffree-form
```

4.22 -fhonor-infinities

Do not allow optimizations that assume the arguments and results of floating point arithmetic are not +/- Inf.

Syntax

```
armflang -fhonor-infinities, -fno-honor-infinities
```

4.23 -fhonor-nans

Do not allow optimizations that assume the arguments and results of floating point arithmetic are not NaN.

Syntax

```
armclang -fhonor-nans, -fno-honor-nans
```

4.24 -finline

Enable/disable inlining (enabled by default).

Syntax

```
armflang -finline, -fno-inline
```

4.25 -finline-functions

Inline suitable functions.

Syntax

```
armflang -finline-functions
```

4.26 -finline-hint-functions

Inline functions which are (explicitly or implicitly) marked inline.

Syntax

```
armflang -finline-hint-functions
```

4.27 **-fiterative-reciprocal**

Allow optimizations that replace division by reciprocal estimation and refinement.

Syntax

```
armflang -fiterative-reciprocal, -fno-iterative-reciprocal
```

4.28 -flto

Enable (-flto) or disable (-fno-lto) Link Time Optimizations (LTO).

You must pass the option to both the link and compile commands. When LTO is enabled, compiler object files contain an intermediate representation of the original code. When linking the objects together into a binary at link time, the compiler performs optimizations. It can allow the compiler to inline functions from different files, for example.

Default

Default is -fno-lto.

Syntax

```
armflang -flto, -fno-lto
```


4.29 -fmath-errno

Require math functions to indicate errors.

Use `-fmath-errno` if your source code uses `errno` to check the status of math function calls. If your code never uses `errno`, you can use `-fno-math-errno` to unlock optimizations such as:

1. In C/C++ it allows `sin()` and `cos()` calls that take the same input to be combined into a more efficient `sincos()` call.
2. In C/C++ it allows certain `pow(x, y)` function calls to be eliminated completely when `y` is a small integral value.

Syntax

```
armflang -fmath-errno, -fno-math-errno
```

4.30 -fnative-atomics

Enable (-fnative-atomics) or disable (-fno-native-atomics) the use of native atomic instructions for OpenMP atomics. Default is -fnative-atomics.

By default, armflang generates native atomic instructions for OpenMP atomic operations, falling back to libatomic when no suitable native instruction is available. Use -fno-native-atomics to disable this feature and have armflang generate code that use barriers to guarantee atomicity. This will normally result in a slower program.

Default

Default is -fnative-atomics.

Syntax

```
armflang -fnative-atomics, -fno-native-atomics
```

4.31 -fno-crash-diagnostics

Disable the auto-generation of preprocessed source files and a script for reproduction during a clang crash.

Syntax

```
armclang -fno-crash-diagnostics
```

4.32 -fno-fortran-main

Do not link in Fortran main.

Syntax

```
armflang -fno-fortran-main
```

4.33 -fopenmp

Enable OpenMP and link in the OpenMP library, libomp.

Syntax

```
armflang -fopenmp
```

4.34 -fopenmp-simd

Enable processing of ‘simd’ and the ‘declare simd’ pragma, without enabling OpenMP or linking in the OpenMP library, libomp. Enabled by default.

Syntax

```
armflang -fopenmp-simd, -fno-openmp-simd
```

4.35 -frealloc-lhs

Select semantics for assignments to allocatables.

Fortran 2003 allows dynamic reallocation, which will error in Fortran 90/95. Use `-fno-realloc-lhs` to restore the F95 behavior. Default is F2003 semantics (`-frealloc-lhs`).

Default

Default is F2003 semantics (`-frealloc-lhs`).

Syntax

```
armflang -frealloc-lhs, -fno-realloc-lhs
```

4.36 -freciprocal-math

Allow division operations to be reassociated

Syntax

```
armflang -freciprocal-math
```


4.37 -frecursive

Allocate all local arrays on the stack, allowing thread-safe recursion (enabled by default with -fopenmp).

In the absence of this flag, some large local arrays may be allocated in static memory. This reduces stack usage, but is not thread-safe.

Default

-frecursive is enabled by default with -fopenmp.

Syntax

```
armflang -frecursive
```

4.38 -fsave-optimization-record

Enable (-fsave-optimization-record) or disable (-fno-save-optimization-record) the generation of a YAML optimization record file. Default is -fno-save-optimization-record.

Optimization records are files named <output name>.opt.yaml, which can be parsed by arm-opt-report to show what optimization decisions the compiler is making, in-line with your source code. For more information, see the ‘Optimize’ chapter in the compiler developer and reference guide.

Default

Default is fno-save-optimization-record.

Syntax

```
armflang -fsave-optimization-record, -fno-save-optimization-record
```

4.39 -fsign-zero

Write floating point numbers of the value zero with the sign bit as a negative number. Enabled by default.

Also write the numbers in formatted output and treat them as negative when used by the ‘SIGN’ intrinsic.

Syntax

```
armflang -fsign-zero, -fno-sign-zero
```

4.40 -fsigned-char

Set the type of 'char' to be signed. Disabled by default.

Syntax

```
armflang -fsigned-char, -fno-signed-char
```

4.41 -fsigned-zeros

Do not allow optimizations that ignore the sign of floating point zeros. Enabled by default.

Syntax

```
armflang -fsigned-zeros, -fno-signed-zeros
```

4.42 -fsimdmath

Enables the vectorized libm library to support the vectorization of loops containing calls to basic library functions, such as those declared in `math.h`

Syntax

```
armflang -fsimdmath, -fno-simdmath
```

4.43 -fstack-arrays

Place all automatic arrays on stack memory (enabled by default with -Ofast).

Use this option if your Fortran code frequently performs small allocations and deallocations of memory. -fstack-arrays improves application performance by using memory on the stack instead of allocating it through malloc, or similar. For programs using very large arrays on particular operating systems, consider extending stack memory runtime limits.

Syntax

```
armflang -fstack-arrays, -fno-stack-arrays
```

4.44 -fstrict-aliasing

Tells the compiler to adhere to the aliasing rules defined in the source language (enabled by default when using ‘-Ofast’).

In some circumstances, this flag allows the compiler to assume that pointers to different types do not alias.

Syntax

```
armflang -fstrict-aliasing
```


4.45 -fsyntax-only

Show syntax errors but do not perform any compilation.

Syntax

```
armflang -fsyntax-only
```

4.46 -ftrapping-math

Tell the compiler to assume (-ftrapping-math), or not to assume (-fno-trapping-math), that floating point operations can trap.

Possible traps include:

- Division by zero
- Underflow
- Overflow
- Inexact result
- Invalid operation.

Syntax

```
armflang -ftrapping-math, -fno-trapping-math
```

4.47 -funsafe-math-optimizations

Enable a subset of the optimizations allowed by -ffast-math.

Using --funsafe-math-optimizations is equivalent to specifying the following flags individually:

-fassociative-math, -freciprocal-math, -fno-signed-zeros, and -fno-trapping-math

Syntax

```
armclang -funsafe-math-optimizations, -fno-unsafe-math-optimizations
```

4.48 -fvectorize

Enable/disable loop vectorization (enabled by default).

Syntax

```
armflang -fvectorize, -fno-vectorize
```

4.49 -g

Generate source-level debug information.

Syntax

```
armclang -g
```

4.50 -g0

Disable generation of source-level debug information (default).

Syntax

```
armflang -g0
```

4.51 -gcc-toolchain=

Use the gcc toolchain at the given directory.

Syntax

```
armflang --gcc-toolchain=<arg>
```

4.52 -gline-tables-only

Emit debug line number tables only.

Syntax

```
armflang -gline-tables-only
```


4.53 -help

Display available options.

Syntax

```
armflang -help, --help
```

4.54 -help-hidden

Display hidden options. Only use these options if advised to do so by your Arm representative.

Syntax

```
armflang --help-hidden
```

4.55 -I

Add directory to include search path. Directories specified with the -I option apply to both the quote form of the include directive and the system header form.

For example, #include “file” (quote form), and #include <file> (system header form).

Syntax

```
armflang -I<dir>
```

4.56 -i8

Treat INTEGER and LOGICAL as INTEGER*8 and LOGICAL*8.

Syntax

```
armflang -i8
```

4.57 -include

Include file before parsing.

Syntax

```
armflang -include<file>, --include<file>
```

4.58 -iquote

Add directory to the include search path.

Directories specified with the `-iquote` option only apply to the quote form of the include directive, such as `#include "file"`.

Syntax

```
armflang -iquote<directory>
```

4.59 -isysroot

For header files, set the system root directory (usually /).

Syntax

```
armflang -isysroot<dir>
```

4.60 -isystem

Override the system include directory.

Directories specified with the `-isystem` option apply to both the quote form of the include directive, such as `#include "file"`, and the system header form, such as `#include <file>`.

Directories specified with this option will be searched after directories specified by the `-iquote` or `-I` options, but before the standard system include directory.

Syntax

```
armflang -isystem<directory>
```


4.61 -isystem-after

Add directory to end of the SYSTEM include search path.

Syntax

```
armflang -isystem-after<directory>
```

4.62 -l

Search for the library named <library> when linking.

Syntax

```
armflang -l<library>
```

4.63 -Mnomain

Do not link in Fortran main.

Syntax

```
armflang -Mnomain
```

4.64 -Mstandard

Check Fortran standard conformance.

Syntax

```
armflang -Mstandard
```

4.65 -march=

Specifies the base architecture and extensions available on the target.

Usage: -march=<arg> where <arg> is constructed as *name*[+*[no]*feature+...]:

name

armv8-a : Armv8 application architecture profile.

armv8.1-a : Armv8.1 application architecture profile.

armv8.2-a : Armv8.2 application architecture profile.

feature

Is the name of an optional architectural feature that can be explicitly enabled with +feature and disabled with +nofeature.

For AArch64, the following features can be specified:

- **crc** - Enable CRC extension. On by default for -march=armv8.1-a or higher.
- **crypto** - Enable Cryptographic extension.
- **fullfp16** - Enable FP16 extension.
- **lse** - Enable Large System Extension instructions. On by default for -march=armv8.1-a or higher.
- **sve** - Scalable Vector Extension (SVE). This feature also enables fullfp16. See [Scalable Vector Extension](#) for more information.
- **sve2** - Scalable Vector Extension version two (SVE2). This feature also enables sve. See [Arm A64 Instruction Set Architecture](#) for SVE and SVE2 instructions.
- **sve2-aes** - SVE2 Cryptographic extension. This feature also enables sve2.
- **sve2-bitperm** - SVE2 Cryptographic Extension. This feature also enables sve2.
- **sve2-sha3** - SVE2 Cryptographic Extension. This feature also enables sve2.
- **sve2-sm4** - SVE2 Cryptographic Extension. This feature also enables sve2.

Note

When enabling either the sve2 or sve features, to link to the SVE-enabled version of Arm Performance Libraries, you must also include the -armpl=sve option. For more information about the supported options for -armpl, see the -armpl description.

Syntax

```
armflang -march=<arg>
```

4.66 -mcpu=

Select which CPU architecture to optimize for.

Syntax

```
armclang -mcpu=<arg>
```

Arguments

native

Auto-detect the CPU architecture from the build computer.

thunderx2t99

Optimize for Marvell ThunderX2 based computers.

neoverse-n1

Optimize for Neoverse N1 based computers.

a64fx

Optimize for Fujitsu A64FX based computers.

generic

Generate portable output suitable for any Armv8-A based computer.

4.67 -module

Fortran module path.

Syntax

```
armflang -module <arg>
```

4.68 -no-flang-libs

Do not link against Flang libraries.

Syntax

```
armflang -no-flang-libs
```


4.69 -nocpp

Do not preprocess Fortran files.

Syntax

```
armflang -nocpp
```

4.70 -O

Specifies the level of optimization to use when compiling source files.

Syntax

```
armclang -O<level>
```

Arguments

0

Minimum optimization for the performance of the compiled binary. Turns off most optimizations. When debugging is enabled, this option generates code that directly corresponds to the source code. Therefore, this might result in a significantly larger image. This is the default optimization level.

1

Restricted optimization. When debugging is enabled, this option gives the best debug view for the trade-off between image size, performance, and debug.

2

High optimization. When debugging is enabled, the debug view might be less satisfactory because the mapping of object code to source code is not always clear. The compiler might perform optimizations that cannot be described by debug information.

3

Very high optimization. When debugging is enabled, this option typically gives a poor debug view. Arm recommends debugging at lower optimization levels.

fast

Enables all the optimizations from level 3 including those performed with the `-ffp-mode=fast` `armclang` option. This level also performs other aggressive optimizations that might violate strict compliance with language standards. `-Ofast` implies `-ffast-math`.

4.71 -o

Write output to <file>.

Syntax

```
armflang -o<file>
```

4.72 -print-search-dirs

Print the paths that are used for finding libraries and programs.

Syntax

```
armflang -print-search-dirs, --print-search-dirs
```

4.73 -Qunused-arguments

Do not emit a warning for unused driver arguments.

Syntax

```
armflang -Qunused-arguments
```

4.74 -r8

Treat REAL as REAL*8.

Syntax

```
armflang -r8
```

4.75 -S

Only run preprocess and compilation steps.

Syntax

```
armclang -S
```

4.76 -shared

Create a shared object that can be linked against.

Syntax

```
armflang -shared, --shared
```


4.77 -static

Link against static libraries.

This option prevents runtime dependencies on shared libraries. This is likely to result in larger binaries.

Syntax

```
armflang -static, --static
```

4.78 -static-arm-libs

Link against static Arm libraries.

This option prevents runtime dependencies on libraries shipped with Arm Compiler for Linux (such as libamath, libastring and Arm Performance Libraries). This is likely to result in larger binaries.

Syntax

```
armflang -static-arm-libs
```

4.79 -std=

Language standard to compile for.

The list of valid standards depends on the input language, but adding `-std=` to a build line will generate an error message listing valid choices.

Syntax

```
armflang -std=<arg>, --std=<arg>
```

4.80 -U

Undefine macro <macro>.

Syntax

```
armclang -U<macro>
```

4.81 -v

Show commands to run and use verbose output.

Syntax

```
armflang -v
```

4.82 -version

Show the version number and some other basic information about the compiler.

Syntax

```
armflang --version, --vsn
```

4.83 -W

Enable the specified warning.

Similarly, warnings can be disabled with `-Wno-<warning>`.

Syntax

```
armflang -W<warning>
```

4.84 -Wall

Enable all warnings.

Syntax

```
armclang -Wall
```


4.85 -Warm-extensions

Enable warnings about the use of non-standard language features supported by armclang or armflang

Syntax

```
armflang -Warm-extensions
```

4.86 -Wdeprecated

Enable warnings for deprecated constructs and define `__DEPRECATED`.

Syntax

```
armclang -Wdeprecated
```

4.87 -Wl,

Pass the comma separated arguments in <arg> to the linker.

Syntax

```
armclang -Wl,<arg>,<arg2>...
```

4.88 -Wp,

Pass the comma separated arguments in <arg> to the preprocessor

Syntax

```
armclang -Wp,<arg>,<arg2>...
```

4.89 -w

Suppress all warnings.

Syntax

```
armflang -w
```

4.90 -working-directory

Resolve file paths relative to the specified directory.

Syntax

```
armflang -working-directory<arg>
```

4.91 -Xassembler

Pass <arg> to the assembler.

Syntax

```
armflang -Xassembler <arg>
```

4.92 -Xlinker

Pass <arg> to the linker.

Syntax

```
armflang -Xlinker <arg>
```


4.93 -Xpreprocessor

Pass <arg> to the preprocessor.

Syntax

```
armclang -Xpreprocessor <arg>
```

Chapter 5

Fortran language reference

Arm Fortran Compiler supports Fortran 2008, Fortran 2003, Fortran 95, Fortran 90, and some Fortran 2018 language features. This chapter can be used as a reference for the supported language features.

The support level for the latest Fortran standards (2003 and 2008) are described in [Standards support on page 6-169](#).

For information about the Fortran standards, see the [JTC1/SC22/WG5 Fortran standards website](#).

It contains the following sections:

- [5.1 Data types and file extensions](#) on page 5-139.
- [5.2 Ininsics](#) on page 5-144.
- [5.3 Statements](#) on page 5-162.

5.1 Data types and file extensions

Describes the data types and file extensions that are supported by the Arm Fortran Compiler.

This section contains the following subsections:

- [5.1.1 Data types on page 5-139.](#)
- [5.1.2 Supported file extensions on page 5-140.](#)
- [5.1.3 Logical variables and constants on page 5-141.](#)
- [5.1.4 C/Fortran inter-language calling on page 5-141.](#)
- [5.1.5 Character on page 5-142.](#)
- [5.1.6 Complex on page 5-142.](#)
- [5.1.7 Fortran implementation notes on page 5-142.](#)

5.1.1 Data types

Arm Fortran Compiler provides the following intrinsic data types:

Table 5-1 Intrinsic data types

Data Type	Specified as	Size (bytes)
INTEGER	INTEGER	4
	INTEGER*1	1
	INTEGER([KIND=]1)	1
	INTEGER*2	2
	INTEGER([KIND=]2)	2
	INTEGER*4	4
	INTEGER([KIND=]4)	4
	INTEGER*8	8
	INTEGER([KIND=]8)	8
REAL	REAL	4
	REAL*4	4
	REAL([KIND=]4)	4
	REAL*8	8
	REAL([KIND=]8)	8
DOUBLE PRECISION	DOUBLE PRECISION (same as REAL*8, no KIND parameter is permitted)	16
COMPLEX	COMPLEX	4
	COMPLEX*8	8
	COMPLEX([KIND=]4)	8
	COMPLEX*16	16
	COMPLEX([KIND=]8)	16
DOUBLE COMPLEX	DOUBLE COMPLEX (same as COMPLEX*8, no KIND parameter is permitted)	8

Table 5-1 Intrinsic data types (continued)

Data Type	Specified as	Size (bytes)
LOGICAL	LOGICAL	4
	LOGICAL*1	1
	LOGICAL([KIND=]1)	1
	LOGICAL*2	2
	LOGICAL([KIND=]2)	2
	LOGICAL*4	4
	LOGICAL([KIND=]4)	4
	LOGICAL*8	8
	LOGICAL([KIND=]8)	8
CHARACTER	CHARACTER	1
	CHARACTER([KIND=]1)	1
BYTE	BYTE (same as INTEGER([KIND=]1))	1

Note

- The default entries are the first entries for each intrinsic data type.
- To determine the kind type parameter of a representation method, use the intrinsic function `KIND`.

For more portable programs, define a `PARAMETER` constant using the appropriate `SELECTED_INT_KIND` or `SELECTED_REAL_KIND` functions, as appropriate.

For example, this code defines a `PARAMETER` constant for an `INTEGER` kind that has 9 digits:

```
INTEGER, PARAMETER :: MY_INT_KIND = SELECTED_INT_KIND(9)
...
INTEGER(MY_INT_KIND) :: J
...
```

5.1.2 Supported file extensions

The extensions `.f90`, `.f95`, `.f03`, and `.f08` are used for modern, free-form source code conforming to the Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008 standards, respectively.

The extensions `.F90`, `.F95`, `.F03`, and `.F08` are used for modern, free-form source code that require preprocessing, and conform to the Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008 standards, respectively.

The `.f` and `.for` extensions are typically used for older, fixed-form code such as `FORTRAN77`.

The file extensions that are compatible with Arm Fortran Compiler are:

Table 5-2 Supported file extensions

File Extension	Interpretation
a.out	Executable output file.
file.a	Library of object files.

Table 5-2 Supported file extensions (continued)

File Extension	Interpretation
file.f file.for	Fixed-format Fortran source file.
file.fpp file.F	Fixed-format Fortran source file that requires preprocessing.
file.f90 file.f95 file.f03 file.f08	Free-format Fortran source file.
file.F90 file.F95 file.F03 file.F08	Free-format Fortran source file that requires preprocessing.
file.o	Compiled object file.
file.s	Assembler source file.

5.1.3 Logical variables and constants

This topic describes LOGICAL variables and constants.

A LOGICAL constant is either `True` or `False`. The Fortran standard does not specify how variables of LOGICAL type are represented. However, it does require LOGICAL variables of default kind to have the same storage size as default INTEGER and REAL variables.

For Arm Fortran Compiler:

- `.TRUE.` corresponds to -1 and has a default storage size of 4-bytes.
- `.FALSE.` corresponds to 0 and has a default storage size of 4-bytes.

Note

Some compilers represent `.TRUE.` and `.FALSE.` as 1 and 0, respectively.

5.1.4 C/Fortran inter-language calling

This section provides some useful troubleshooting information when handling argument passing and return values for Fortran functions or subroutines that are called from C/C++ code.

In Fortran, arguments are passed by reference. Here, reference means the address of the argument is passed, rather than the argument itself. In C/C++, arguments are passed by value, except for strings and arrays, which are passed by reference.

C/C++ provides some flexibility when solving passing difference with Fortran. Usually, intelligent use of the `&` and `*` operators in argument passing enables you to call Fortran from C/C++, and in argument declarations when Fortran is calling C/C++.

Fortran functions which return CHARACTER or COMPLEX data types require special consideration when called from C/C++ code.

5.1.5 Character

This topic describes how C/C++ functions call Fortran functions that return a CHARACTER.

Fortran functions that return a CHARACTER require the *calling* C/C++ function to have two arguments to describe the result:

1. The first argument provides the address of the returned character.
2. The second argument provides the length of the returned character.

For example, the Fortran function:

```
CHARACTER*(*) FUNCTION CHF( C1, I)
  CHARACTER*(*) C1
  INTEGER I
END
```

when called in C/C++, has an extra declaration:

```
extern void chf_();
char tmp[10];
char c1[9];
int i;
chf_(tmp, 10, c1, &i, 9);
```

The argument, tmp, provides the address, and the length is defined with the second argument, 10.

————— **Note** —————

- Fortran functions declared with a character return length, for example CHARACTER*4 FUNCTION CHF(), still require the second parameter to be supplied to the calling C/C++ code.
- The value of the character function is not automatically NULL-terminated.

5.1.6 Complex

This topic describes how to call Fortran functions that return a COMPLEX data type, from C or C++.

Fortran functions that return a COMPLEX data type cannot be directly called from C or C++. Instead, a workaround is possible by passing a C or C++ function a pointer to a memory area. This memory area can then be calling the COMPLEX function and storing the returned value.

For example, the Fortran function:

```
SUBROUTINE INTER_CF(C, I)
  COMPLEX C
  COMPLEX CF
  C = CF(I)
  RETURN
END
COMPLEX FUNCTION CF(I)
  . . .
END
```

when called in C/C++ is completed using a memory pointer:

```
extern void inter_cf_();
typedef struct {float real, imag;} cplx;
cplx c1;
int i;
inter_cf_( &c1, &i);
```

5.1.7 Fortran implementation notes

Details information that is specific to the implementation of Fortran in Arm Fortran Compiler.

Implementation information:

- **Arm Fortran Compiler does not initialize arrays or variables with zeros.**

Note

This behavior varies from compiler to compiler and is not defined in Fortran standards. The best practice is not to assume that arrays are filled with zeros when they are created.

Related concepts

[5.1.4 C/Fortran inter-language calling](#) on page 5-141

[5.1.6 Complex](#) on page 5-142

Related references

[5.1.1 Data types](#) on page 5-139

[5.1.2 Supported file extensions](#) on page 5-140

[5.1.3 Logical variables and constants](#) on page 5-141

[5.1.5 Character](#) on page 5-142

[5.1.7 Fortran implementation notes](#) on page 5-142

5.2 Intrinsic

The Fortran language standards that are implemented in Arm Fortran Compiler are Fortran 77, Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008. This topic details the supported and unsupported Fortran intrinsics in Arm Fortran Compiler.

This section contains the following subsections:

- [5.2.1 Fortran intrinsics overview on page 5-144.](#)
- [5.2.2 Bit manipulation functions and subroutines on page 5-144.](#)
- [5.2.3 Elemental character and logical functions on page 5-145.](#)
- [5.2.4 Vector/Matrix functions on page 5-147.](#)
- [5.2.5 Array reduction functions on page 5-147.](#)
- [5.2.6 String construction functions on page 5-149.](#)
- [5.2.7 Array construction manipulation functions on page 5-149.](#)
- [5.2.8 General inquiry functions on page 5-150.](#)
- [5.2.9 Numeric inquiry functions on page 5-150.](#)
- [5.2.10 Array inquiry functions on page 5-151.](#)
- [5.2.11 Transfer functions on page 5-151.](#)
- [5.2.12 Arithmetic functions on page 5-152.](#)
- [5.2.13 Miscellaneous functions on page 5-155.](#)
- [5.2.14 Subroutines on page 5-155.](#)
- [5.2.15 Fortran 2003 functions on page 5-156.](#)
- [5.2.16 Fortran 2008 functions on page 5-156.](#)
- [5.2.17 Unsupported functions on page 5-158.](#)
- [5.2.18 Unsupported subroutines on page 5-160.](#)

5.2.1 Fortran intrinsics overview

An intrinsic is a function made available for a given language standard, for example, Fortran 95. Intrinsic functions accept arguments and return values. When an intrinsic function is called in the source code, the compiler replaces the function with a set of automatically generated instructions. It is best practice to use these intrinsics to enable the compiler to optimize the code most efficiently.

Note

The intrinsics listed in the following tables are specific to Fortran 90/95, unless explicitly stated.

5.2.2 Bit manipulation functions and subroutines

Functions and subroutines for manipulating bits.

Table 5-3 Bit manipulation functions and subroutines

Intrinsic	Description	Num. of Arguments	Argument Type	Result
AND	Perform a logical AND on corresponding bits of the arguments.	2	Any, except CHAR or COMPLEX	INTEGER or LOGICAL
BIT_SIZE	Return the number of bits (the precision) of the integer argument.	1	INTEGER	INTEGER
BTEST	Test the binary value of a bit in a specified position of an integer argument.	2	INTEGER, INTEGER	LOGICAL
IAND	Perform a bit-by-bit logical AND on the arguments.	2	INTEGER, INTEGER (of same kind)	INTEGER
IBCLR	Clear one bit to zero.	2	INTEGER, INTEGER ≥ 0	INTEGER

Table 5-3 Bit manipulation functions and subroutines (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
IBITS	Extract a sequence of bits.	3	INTEGER, INTEGER ≥ 0 , INTEGER ≥ 0	INTEGER
IBSET	Set one bit to one.	2	INTEGER, INTEGER ≥ 0	INTEGER
IEOR	Perform a bit-by-bit logical exclusive OR on the arguments.	2	INTEGER, INTEGER (of same kind)	INTEGER
IOR	Perform a bit-by-bit logical OR on the arguments.	2	INTEGER, INTEGER (of same kind)	INTEGER
ISHFT	Perform a logical shift.	2	INTEGER, INTEGER	INTEGER
ISHFTC	Perform a circular shift of the rightmost bits.	2 or 3	INTEGER, INTEGER or INTEGER, INTEGER, INTEGER	INTEGER
LSHIFT	Perform a logical shift to the left.	2	INTEGER, INTEGER	INTEGER
MVBITS	Copy bit sequence.	5	INTEGER(IN), INTEGER(IN), INTEGER(IN), INTEGER(IN, OUT), INTEGER(IN)	N/A
NOT	Perform a bit-by-bit logical complement on the argument.	2	INTEGER	INTEGER
OR	Perform a logical OR on each bit of the arguments.	2	Any except CHAR or COMPLEX	INTEGER or LOGICAL
POPCNT	Return the number of one bits. (F2008)	1	INTEGER or bits	INTEGER
POPPAR	Return the bitwise parity. (F2008)	1	INTEGER or bits	INTEGER
RSHIFT	Perform a logical shift to the right.	2	INTEGER, INTEGER	INTEGER
SHIFT	Perform a logical shift.	2	Any except CHAR or COMPLEX, INTEGER	INTEGER or LOGICAL
XOR	Perform a logical exclusive OR on each bit of the arguments.	2	INTEGER, INTEGER	INTEGER
ZEXT	Zero-extend the argument.	1	INTEGER or LOGICAL	INTEGER

5.2.3 Elemental character and logical functions

Elemental character logical conversion functions.

Table 5-4 Elemental character and logical functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
ACHAR	Return character in specified ASCII collating position.	1	INTEGER	CHARACTER
ADJUSTL	Left adjust string.	1	CHARACTER	CHARACTER
ADJUSTR	Right adjust string.	1	CHARACTER	CHARACTER

Table 5-4 Elemental character and logical functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
CHAR	Return character with specified ASCII value.	1	LOGICAL*1 INTEGER	CHARACTER CHARACTER
IACHAR	Return position of character in ASCII collating sequence.	1	CHARACTER	INTEGER
ICHAR	Return position of character in the character set's collating sequence.	1	CHARACTER	INTEGER
INDEX	Return starting position of substring in first string.	2 3	CHARACTER, CHARACTER CHARACTER, CHARACTER, LOGICAL	INTEGER INTEGER
LEN	Return the length of string.	1	CHARACTER	INTEGER
LEN_TRIM	Return the length of the supplied string minus the number of trailing blanks.	1	CHARACTER	INTEGER
LGE	Test the supplied strings to determine if the first string is lexically greater than or equal to the second. Note: From F2008, character kind ASCII is also supported.	2	CHARACTER, CHARACTER	LOGICAL
LGT	Test the supplied strings to determine if the first string is lexically greater than the second. Note: From F2008, character kind ASCII is also supported.	2	CHARACTER, CHARACTER	LOGICAL
LLE	Test the supplied strings to determine if the first string is lexically less than or equal to the second. Note: From F2008, character kind ASCII is also supported.	2	CHARACTER, CHARACTER	LOGICAL
LLT	Test the supplied strings to determine if the first string is lexically less than the second. Note: From F2008, character kind ASCII is also supported.	2	CHARACTER, CHARACTER	LOGICAL
LOGICAL	Logical conversion.	1 2	LOGICAL LOGICAL, INTEGER	LOGICAL LOGICAL

Table 5-4 Elemental character and logical functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
SCAN	Scan string for characters in set.	2 3	CHARACTER, CHARACTER CHARACTER, CHARACTER, LOGICAL	INTEGER INTEGER
VERIFY	Determine if string contains all characters in set.	2 3	CHARACTER, CHARACTER CHARACTER, CHARACTER, LOGICAL	INTEGER INTEGER

5.2.4 Vector/Matrix functions

Functions for vector or matrix multiplication.

Table 5-5 Vector and matrix functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
DOT_PRODUCT	Perform dot product on two vectors.	2	INTEGER, REAL, COMPLEX, or LOGICAL	INTEGER, REAL, COMPLEX, or LOGICAL
MATMUL	Perform matrix multiply on two matrices.	2	INTEGER, REAL, COMPLEX, or LOGICAL	INTEGER, REAL, COMPLEX, or LOGICAL

————— **Note** —————

All matrix outputs are the same type as the argument supplied.

—————

5.2.5 Array reduction functions

Functions for determining information from, or calculating using, the elements in an array.

Table 5-6 Array reduction functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
ALL	Determine if all array values are true.	1	LOGICAL	LOGICAL
		2	LOGICAL, INTEGER	LOGICAL
ANY	Determine if any array value is true.	1	LOGICAL	LOGICAL
		2	LOGICAL, INTEGER	LOGICAL
COUNT	Count true values in array.	1	LOGICAL	INTEGER
		2	LOGICAL, INTEGER	INTEGER

Table 5-6 Array reduction functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
MAXLOC	Determine the position of the array element with the maximum value.	1	INTEGER	INTEGER
		2	INTEGER, LOGICAL	INTEGER
		2	INTEGER, INTEGER	INTEGER
		3	INTEGER, INTEGER, LOGICAL	INTEGER
		1	REAL	REAL
		2	REAL, LOGICAL	REAL
		2	REAL, INTEGER	REAL
		3	REAL, INTEGER, LOGICAL	REAL
MAXVAL	Determine the maximum value of the array elements.	1	INTEGER	INTEGER
		2	INTEGER, LOGICAL	INTEGER
		2	INTEGER, INTEGER	INTEGER
		3	INTEGER, INTEGER, LOGICAL	INTEGER
		1	REAL	REAL
		2	REAL, LOGICAL	REAL
		2	REAL, INTEGER	REAL
		3	REAL, INTEGER, LOGICAL	REAL
MINLOC	Determine the position of the array element with the minimum value.	1	INTEGER	INTEGER
		2	INTEGER, LOGICAL	INTEGER
		2	INTEGER, INTEGER	INTEGER
		3	INTEGER, INTEGER, LOGICAL	INTEGER
		1	REAL	REAL
		2	REAL, LOGICAL	REAL
		2	REAL, INTEGER	REAL
		3	REAL, INTEGER, LOGICAL	REAL
MINVAL	Determine the minimum value of the array elements.	1	INTEGER	INTEGER
		2	INTEGER, LOGICAL	INTEGER
		2	INTEGER, INTEGER	INTEGER
		3	INTEGER, INTEGER, LOGICAL	INTEGER
		1	REAL	REAL
		2	REAL, LOGICAL	REAL
		2	REAL, INTEGER	REAL
		3	REAL, INTEGER, LOGICAL	REAL

Table 5-6 Array reduction functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
PRODUCT	Calculate the product of the elements of an array.	1	NUMERIC	NUMERIC
		2	NUMERIC, LOGICAL	NUMERIC
		2	NUMERIC, INTEGER	NUMERIC
		3	NUMERIC, INTEGER, LOGICAL	NUMERIC
SUM	Calculate the sum of the elements of an array.	1	NUMERIC	NUMERIC
		2	NUMERIC, LOGICAL	NUMERIC
		2	NUMERIC, INTEGER	NUMERIC
		3	NUMERIC, INTEGER, LOGICAL	NUMERIC

5.2.6 String construction functions

Functions for constructing strings.

Table 5-7 String construction functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
REPEAT	Concatenate copies of a string.	2	CHARACTER, INTEGER	CHARACTER
TRIM	Remove trailing blanks from a string.	1	CHARACTER	CHARACTER

5.2.7 Array construction manipulation functions

Functions for constructing and manipulating arrays.

Table 5-8 Array construction and manipulation functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
CSHIFT	Perform circular shift on an array.	2	ARRAY, INTEGER	ARRAY
		3	ARRAY, INTEGER, INTEGER	ARRAY
OESHIFT	Perform end-off shift on an array.	2	ARRAY, INTEGER	ARRAY
		3	ARRAY, INTEGER, Any	ARRAY
		3	ARRAY, INTEGER, INTEGER	ARRAY
		4	ARRAY, INTEGER, Any, INTEGER	ARRAY, ARRAY
MERGE	Merge two arguments using the logical mask.	3	Any, Any, LOGICAL The second argument must be of the same type as the first argument.	Any
PACK	Pack an array into a rank-one array.	2	ARRAY, LOGICAL	ARRAY
		3	ARRAY, LOGICAL, VECTOR	ARRAY

Table 5-8 Array construction and manipulation functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
RESHIFT	Change the shape of an array.	2	ARRAY, INTEGER	ARRAY
		3	ARRAY, INTEGER, ARRAY	ARRAY
		3	ARRAY, INTEGER, INTEGER	ARRAY
		4	ARRAY, INTEGER, ARRAY, INTEGER	ARRAY
SPREAD	Replicate an array by adding a dimension.	3	Any, INTEGER, INTEGER	ARRAY
TRANSPOSE	Transpose an array of rank two.	1	ARRAY (m, n)	ARRAY (n, m)
UNPACK	Unpack a rank-one array into an array of multiple dimensions.	3	VECTOR, LOGICAL, ARRAY	ARRAY

Note

All ARRAY outputs are the same type as the argument supplied.

5.2.8 General inquiry functions

Functions for general determining.

Table 5-9 General inquiry functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
ASSOCIATED	Determine association status.	1	POINTER, POINTER, ..., POINTER, TARGET	LOGICAL
		2		LOGICAL
KIND	Determine the kind of an argument.	1	Any intrinsic type	INTEGER
PRESENT	Determine presence of optional argument.	1	Any	LOGICAL

5.2.9 Numeric inquiry functions

Functions for determining numeric information.

Table 5-10 Numeric inquiry functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
DIGITS	Determine the number of significant digits.	1	INTEGERREAL	INTEGER
EPSILON	Smallest number that can be represented.	1	REAL	REAL
HUGE	Largest number that can be represented.	1	INTEGERREAL	INTEGERREAL
MAXEXPONENT	Value of the maximum exponent.	1	REAL	INTEGER

Table 5-10 Numeric inquiry functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
MINEXPONENT	Value of the minimum exponent.	1	REAL	INTEGER
PRECISION	Decimal precision.	11	REALCOMPLEX	INTEGER INTEGER
RADIX	Base of the model.	11	INTEGERREAL	INTEGER INTEGER
RANGE	Decimal exponent range.	111	INTEGERREALCOMPLEX	INTEGERINTEGERINTEGER
SELECTED_ INT_KIND	Kind-type titlemeter in range.	1	INTEGER	INTEGER
SELECTED_ REAL_KIND	Kind-type titlemeter in range. Syntax: SELECTED _REAL_KIND(P [,R]) where P is precision and R is the range.	1 2	INTEGER INTEGER, INTEGER	INTEGER INTEGER
TINY	Smallest positive number that can be represented.	1	REAL	REAL

5.2.10 Array inquiry functions

Functions for determining information about an array.

Table 5-11 Array inquiry functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
ALLOCATED	Determine if an array is allocated.	1	ARRAY	LOGICAL
LBOUND	Determine the lower bounds.	1 2	ARRAY ARRAY, INTEGER	INTEGER
SHAPE	Determine the shape.	1	Any	INTEGER
SIZE	Determine the number of elements.	1 2	ARRAY ARRAY, INTEGER	INTEGER
UBOUND	Determine the upper bounds.	1 2	ARRAY ARRAY, INTEGER	INTEGER

5.2.11 Transfer functions

Functions for transferring types.

Table 5-12 Transfer functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
TRANSFER	Change the type but maintain bit representation.	2 3	Any, Any Any, Any, INTEGER	Any*

*Must be of the same type as the second argument

5.2.12 Arithmetic functions

Functions for manipulating arithmetic.

Table 5-13 Arithmetic functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
ABS	Return absolute value of the supplied argument.	1	INTEGER, REAL, or COMPLEX	INTEGER, REAL, or COMPLEX
ACOS	Return the arccosine (in radians) of the specified value.	1	REAL	REAL
ACOSD	Return the arccosine (in degrees) of the specified value.	1	REAL	REAL
AIMAG	Return the value of the imaginary part of a complex number.	1	COMPLEX	REAL
AINT	Truncate the supplied value to a whole number.	2	REAL INTEGER	REAL
AND	Perform a logical AND on corresponding bits of the arguments.	2	Any, except CHAR or COMPLEX	INTEGER or LOGICAL
ANINT	Return the nearest whole number to the supplied argument.	2	REAL, INTEGER	REAL
ASIN	Return the arcsine (in radians) of the specified value.	1	REAL	REAL
ASIND	Return the arcsine (in degrees) of the specified value.	1	REAL	REAL
ATAN	Return the arctangent (in radians) of the specified value.	1	REAL	REAL
ATAN2	Return the arctangent (in radians) of the specified pair of values.	2	REAL, REAL	REAL
ATAN2D	Return the arctangent (in degrees) of the specified pair of values.	1	REAL, REAL	REAL
ATAND	Return the arctangent (in degrees) of the specified value.	1	REAL	REAL
CEILING	Return the least integer greater than or equal to the supplied real argument.	2	REAL, KIND	INTEGER
CMPLX	Convert the supplied argument or arguments to complex type.	2 3	{INTEGER, REAL, or COMPLEX}, {INTEGER, REAL, or COMPLEX} {INTEGER, REAL, or COMPLEX}, {INTEGER or REAL}, KIND	COMPLEX COMPLEX
COMPL	Perform a logical complement on the argument.	1	Any, except CHAR or COMPLEX	N/A

Table 5-13 Arithmetic functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
COS	Return the cosine (in radians) of the specified value.	1	REAL COMPLEX	REAL
COSD	Return the cosine (in degrees) of the specified value.	1	REAL COMPLEX	REAL
COSH	Return the hyperbolic cosine of the specified value.	1	REAL	REAL
DBLE	Convert to double precision real.	1	INTEGER, REAL, or COMPLEX	REAL
DCMPLX	Convert the argument or supplied arguments to double complex type.	1 2	INTEGER, REAL, or COMPLEX INTEGER, REAL	DOUBLE COMPLEX DOUBLE COMPLEX
DPROD	Double precision real product.	2	REAL, REAL	REAL (double precision)
EQV	Perform a logical exclusive NOR on the arguments.	2	Any, except CHAR or COMPLEX	INTEGER or LOGICAL
EXP	Exponential function.	1	REAL COMPLEX	REAL COMPLEX
EXPONENT	Return the exponent part of a real number.	1	REAL	INTEGER
FLOOR	Return the greatest integer less than or equal to the supplied real argument.	1 2	REAL REAL, KIND	REAL KIND
FRACTION	Return the fractional part of a real number.	1	REAL	INTEGER
IINT	Convert a value to a short integer type.	1	INTEGER, REAL, or COMPLEX	INTEGER
ININT	Return the nearest short integer to the real argument.	1	REAL	INTEGER
INT	Convert a value to integer type.	1 2	INTEGER, REAL, or COMPLEX {INTEGER, REAL, or COMPLEX}, KIND	INTEGER INTEGER
INT8	Convert a real value to a long integer type.	1	REAL	INTEGER
IZEXT	Zero-extend the argument.	1	LOGICAL or INTEGER	INTEGER
JINT	Convert a value to an integer type.	1	INTEGER, REAL, or COMPLEX	INTEGER
JNINT	Return the nearest integer to the real argument.	1	REAL	INTEGER
KNINT	Return the nearest integer to the real argument.	1	REAL	INTEGER (long)

Table 5-13 Arithmetic functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
LOG	Return the natural logarithm.	1	REAL or COMPLEX	REAL
LOG10	Return the common logarithm.	1	REAL	REAL
MAX	Return the maximum value of the supplied arguments.	2 or more	INTEGER or REAL (all of same kind)	Same as argument type
MIN	Return the minimum value of the supplied arguments.	2 or more	INTEGER or REAL (all of same kind)	Same as argument type
MOD	Find the remainder.	2 or more	{INTEGER or REAL}, {INTEGER or REAL} (all of same kind)	Same as argument type
MODULO	Return the modulo value of the arguments.	2 or more	{INTEGER or REAL}, {INTEGER or REAL} (all of same kind)	Same as argument type
NEAREST	Return the nearest different number that can be represented, by a machine, in a given direction.	2	REAL, REAL (nonzero)	REAL
NEQV	Perform a logical exclusive OR on the arguments.	2	Any, except CHAR or COMPLEX	INTEGER or LOGICAL
NINT	Convert a value to integer type.	1 2	REAL REAL, KIND	INTEGER
REAL	Convert the argument to real.	1 2	INTEGER, REAL, or COMPLEX {INTEGER, REAL, or COMPLEX}, KIND	REAL REAL
RRSPACING	Return the reciprocal of the relative spacing of model numbers near the argument value.	1	REAL	REAL
SET_EXPONENT	Return the model number whose fractional part is the fractional part of the model representation of the first argument and whose exponent part is the second argument.	2	REAL, INTEGER	REAL
SIGN	Return the absolute value of A times the sign of B. Syntax: SIGN(A, B)	2	{INTEGER or REAL}, {INTEGER or REAL}	Same as argument
SIN	Return the sine (in radians) of the specified value.	1	REAL or COMPLEX	REAL
SIND	Return the sine (in degrees) of the specified value.	1	REAL or COMPLEX	REAL
SINH	Return the hyperbolic sine of the specified value.	1	REAL	REAL
SPACING	Return the relative spacing of model numbers near the argument value.	1	REAL	REAL

Table 5-13 Arithmetic functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
SQRT	Return the square root of the argument.	1	REAL or COMPLEX	REAL or COMPLEX
TAN	Return the tangent (in radians) of the specified value.	1	REAL	REAL
TAND	Return the tangent (in degrees) of the specified value.	1	REAL	REAL
TANH	Return the hyperbolic tangent of the specified value.	1	REAL	REAL

5.2.13 Miscellaneous functions

Functions for miscellaneous use.

Table 5-14 Miscellaneous functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
LOC	Return the argument address.	1	NUMERIC	INTEGER
NULL	Assign a disassociated status.	0 1	POINTER	POINTER POINTER

5.2.14 Subroutines

Supported subroutines.

Table 5-15 Subroutines

Intrinsic	Description	Num. of Arguments	Argument Type
CPU_TIME	Return processor time.	1	REAL (OUT)
DATE_AND_TIME	Return the date and time.	4 (all optional)	DATE (CHARACTER, OUT) TIME (CHARACTER, OUT) ZONE (CHARACTER, OUT) VALUES (INTEGER, OUT)
RANDOM_NUMBER	Generate pseudo-random numbers.	1	REAL (OUT)
RANDOM_SEED	Set or query pseudo-random number generator.	1 1 1	SIZE (INTEGER, OUT) PUT (INTEGER ARRAY, IN) GET (INTEGER ARRAY, OUT)
SYSTEM_CLOCK	Query the real time clock.	3 (optional)	COUNT (INTEGER, OUT) COUNT_RATE (REAL, OUT) COUNT_MAX (INTEGER, OUT)

5.2.15 Fortran 2003 functions

Fortran 2003-supported functions.

Table 5-16 Fortran 2003 functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
COMMAND _ARGUMENT _COUNT	Return a scalar of type default integer that is equal to the number of arguments that are passed on the command line when the containing program was invoked. If no command arguments are passed, the result is 0.	0	None	INTEGER
EXTENDS_TYPE _OF	Determine whether the dynamic type of A is an extension type of the dynamic type of B. Syntax: EXTENDS_TYPE _OF (A, B)	2	Objects of extensible type	LOGICAL SCALAR
GET_COMMAND _ARGUMENT	Return the specified command line argument of the command that invoked the program.	1 to 4	INTEGER plus optionally: CHAR, INTEGER, INTEGER	A command argument
GET_COMMAND	Return the entire command line that was used to invoke the program.	0 to 3	CHAR, INTEGER, INTEGER	A command line
GET_ENVIRONM ENT_VARIABLE	Return the value of the specified environment variable.	1 to 5	CHAR, CHAR, INTEGER, INTEGER, LOGICAL	Stores the value of NAME in VALUE
IS_IOSTAT _END	Test whether a variable has the value of the I/O status: 'end of file'.	1	INTEGER	LOGICAL
IS_IOSTAT _EOR	Test whether a variable has the value of the I/O status: 'end of record'.	1	INTEGER	LOGICAL
LEADZ	Count the number of leading zero bits.	1	INTEGER or bits	INTEGER
MOVE_ALLOC	Move an allocation from one allocatable object to another.	2	Any type and rank	None
NEW_LINE	Return the newline character.	1	CHARACTER	CHARACTER
SAME_TYPE _AS	Determine whether the dynamic type of A is the same as the dynamic type of B. Syntax: SAME_TYPE_AS (A, B)	2	Objects of extensible type	LOGICAL SCALAR
SCALE	Return the value $A * B$ where B is the base of the number system in use for A. Syntax: `` SCALE(A, B)``	2	REAL, INTEGER	REAL

5.2.16 Fortran 2008 functions

Fortran 2008-supported functions.

Table 5-17 Fortran 2008 functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
ACOSH ASINH ATANH	Inverse hyperbolic trigonometric functions	1	REAL	REAL
BESSEL_J0 BESSEL_J1 BESSEL_JN BESSEL_Y0 BESSEL_Y1 BESSEL_YN	Bessel function of: (J0) the first kind of order 0. (J1) the first kind of order 1. (JN) the first kind. (Y0) the second kind of order 0. (Y1) the second kind of order 1. (YN) the second kind.	1 1 2 or 3 1 1 2 or 3	REAL REAL {INTEGER, REAL, or INTEGER}, INTEGER, REAL REAL REAL {INTEGER, REAL, or INTEGER}, INTEGER, REAL	REAL REAL REAL REAL REAL REAL
C_SIZEOF	Calculates the number of bytes of storage the expression A ‘occupies’. Syntax: C_SIZEOF(A)	1	Any	INTEGER
COMPILER _OPTIONS	Options passed to the compiler.	None	None	STRING
COMPILER _VERSION	Compiler version string.	None	None	CHARACTER
ERF	Error function.	1	REAL	REAL
ERFC	Complementary error function.	1	REAL	REAL
ERFC _SCALED	Exponentially-scaled complementary error function.	1	REAL	REAL
FINDLOC	Finds the location of a specified value in an array. Syntax: FINDLOC(ARRAY, VALUE, DIM, MASK, KIND, BACK) Or FINDLOC(ARRAY, VALUE, MASK, KIND, BACK)	3 to 6	ARRAY VALUE, DIM[, MASK, KIND, BACK] Or ARRAY, VALUE[, MASK, KIND, BACK]	INTEGER ARRAY
GAMMA	Computes Gamma of A. For positive, integer values of X.	1	REAL (not zero or negative)	REAL
LOG_GAMMA	Computes the natural logarithm of the absolute value of the Gamma function.	1	REAL (not zero or negative)	REAL
HYPOT	Euclidean distance function.	2	REAL, REAL	REAL
IS _CONTIGUOUS	Tests the contiguity of an array.	1	ARRAY	LOGICAL

Table 5-17 Fortran 2008 functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
NORM2	<p>Euclidean vector norm.</p> <p>Syntax:</p> <p>NORM2(X[, DIM])</p> <p>Where: * X shall be a REAL ARRAY. * DIM is an INTEGER SCALAR with a value in the range of 1 to n (where n is the rank of X).</p> <p>————— Note —————</p> <p>The current implementation experiences overflow for arguments containing elements whose square is at the boundary value for double-precision floating-point numbers. There is no such overflow for single-precision arguments.</p>	1[, or 2]	REAL ARRAY[, INTEGER SCALAR]	<p>The result is the same type as X.</p> <p>If DIM is not present, the result is SCALAR. If DIM is present, the result has rank n-1 and shape [d1,d2,...,dDIM-1,DIM+1,...,dn], where n is the rank of X, and [d1,d2,...,dn] is the shape of X.</p>
LEADZ	Returns the number of leading zero bits of an integer.	1	INTEGER	INTEGER
POPCNT	Return the number of one bits.	1	INTEGER	INTEGER
POPPAR	Return the bitwise parity.	1	INTEGER	INTEGER
SELECTED_REAL_KIND	<p>Kind type titlemeter in range.</p> <p>Syntax:</p> <p>SELECTED_REAL_KIND(P[, R, RADIX])</p> <p>where P is precision and R is the range.</p> <p>Note: Radix argument added for F2008.</p>	1 2 3	<p>INTEGER</p> <p>INTEGER, INTEGER</p> <p>INTEGER, INTEGER, INTEGER</p>	<p>INTEGER</p> <p>INTEGER</p> <p>INTEGER</p>
STORAGE_SIZE	<p>Storage size of argument A, in bits.</p> <p>Syntax:</p> <p>STORAGE_SIZE(A[, KIND])</p>	1[, 2]	SCALAR or ARRAY[, INTEGER]	INTEGER
TRAILZ	Number of trailing zero bits of an integer.	1	INTEGER	INTEGER

5.2.17 Unsupported functions

Unsupported Fortran 2008 functions:

Table 5-18 Unsupported functions

Intrinsic	Description	Num. of Arguments	Argument Type	Result
ACOSH ASINH ATANH	Inverse hyperbolic trigonometric functions.	1	COMPLEX	COMPLEX
BGE BGT BLE BLT	Bitwise greater than or equal to. Bitwise greater than. Bitwise less than or equal to. Bitwise less than.	2 2 2 2	INTEGER, INTEGER INTEGER, INTEGER INTEGER, INTEGER INTEGER, INTEGER	LOGICAL LOGICAL LOGICAL LOGICAL
DSHIFTL DSHIFTR	Combined left shift. Combined right shift.	3 3	INTEGER or BOZ constant, INTEGER or BOZ constant, INTEGER INTEGER or BOZ constant, INTEGER or BOZ constant, INTEGER	INTEGER INTEGER
IALL IANY IPARITY	Bitwise AND of array elements. Bitwise OR of array elements. Bitwise XOR of array elements. Syntax: INTRINSIC (ARRAY[, DIM[, MASK]])	1 1 1	ARRAY ARRAY ARRAY	ARRAY ARRAY ARRAY
IMAGE_INDEX NUM_IMAGES THIS_IMAGE	Co-subscript to image index conversion. Number of images. Co-subscript index of this image.	2 0, 1, or 2 0, 1, or 2	COARRAY, INTEGER None, INTEGER, or INTEGER, LOGICAL None, INTEGER, INTEGER or COARRAY, INTEGER	INTEGER INTEGER INTEGER
LCOBOUND UCOBOUND	Lower co-dimension of bounds of an array. Upper co-dimension of bounds of an array. Syntax: INTRINSIC(COARRAY[, DIM[, KIND]])	1 1	COARRAY COARRAY	INTEGER INTEGER
MASKL MASKR	Left justified mask. Right justified mask. Syntax: INTRINSIC(I[, KIND])	1[, or 2] 1[, or 2]	INTEGER[, INTEGER] INTEGER[, INTEGER]	INTEGER INTEGER
MERGE_BITS	Merge of bits under mask.	3	INTEGER, INTEGER, INTEGER	INTEGER

Table 5-18 Unsupported functions (continued)

Intrinsic	Description	Num. of Arguments	Argument Type	Result
PARITY	Reduction with exclusive OR. Syntax: PARITY(MASK[, DIM])	1[, or 2]	LOGICAL ARRAY[,INTEGER]	LOGICAL
SHIFTA	Right shift with fill.	2	INTEGER, INTEGER	INTEGER
SHIFTL	Left shift.	2	INTEGER, INTEGER	INTEGER
SHIFTR	Right shift.	2	INTEGER, INTEGER	INTEGER

5.2.18 Unsupported subroutines

Unsupported Fortran 2008 subroutines:

Table 5-19 Unsupported subroutines

Intrinsic	Description	Num. of Arguments	Argument Type
ATOMIC_DEFINE	Defines the variable ATOM with the value VALUE atomically. Syntax: ATOMIC_DEFINE(ATOM, VALUE[, STAT])	2[, or 3]	{INTEGER or LOGICAL}, {INTEGER or LOGICAL}[, INTEGER]
ATOMIC_REF	Atomically assigns the value of the variable ATOM to VALUE. Syntax: ATOMIC_REF(ATOM, VALUE[, STAT])	2[, or 3]	{INTEGER or LOGICAL}, {INTEGER or LOGICAL}[, INTEGER]
EXECUTE_COMMAND_LINE	Execute a shell command. Syntax: EXECUTE_COMMAND_LINE(COMMAND[, WAIT, EXITSTAT, CMDSTAT, CMDMSG])	1	STRING

Related references

- [5.2.1 Fortran intrinsics overview on page 5-144](#)
- [5.2.2 Bit manipulation functions and subroutines on page 5-144](#)
- [5.2.3 Elemental character and logical functions on page 5-145](#)
- [5.2.4 Vector/Matrix functions on page 5-147](#)
- [5.2.5 Array reduction functions on page 5-147](#)
- [5.2.6 String construction functions on page 5-149](#)
- [5.2.7 Array construction manipulation functions on page 5-149](#)
- [5.2.8 General inquiry functions on page 5-150](#)
- [5.2.9 Numeric inquiry functions on page 5-150](#)
- [5.2.10 Array inquiry functions on page 5-151](#)
- [5.2.11 Transfer functions on page 5-151](#)
- [5.2.12 Arithmetic functions on page 5-152](#)

- [*5.2.13 Miscellaneous functions*](#) on page 5-155
- [*5.2.14 Subroutines*](#) on page 5-155
- [*5.2.15 Fortran 2003 functions*](#) on page 5-156
- [*5.2.16 Fortran 2008 functions*](#) on page 5-156
- [*5.2.17 Unsupported functions*](#) on page 5-158
- [*5.2.18 Unsupported subroutines*](#) on page 5-160

5.3 Statements

Describes the Fortran statements that are supported in[fortrancompiler].

The Fortran statements that are supported in the Arm Fortran Compiler, are:

Table 5-20 Supported Fortran statements

Statement	Language standard	Brief description
ACCEPT	F77	Causes formatted input to be read on standard input.
ALLOCATABLE	F90	Specifies that an array with fixed rank, but deferred shape, is available for a future ALLOCATE statement.
ALLOCATE	F90	Allocates storage for each allocatable array, pointer object, or pointer-based variable that appears in the statements; declares storage for deferred-shape arrays. Note: Arm Fortran Compiler does not initialize arrays or variables with zeros. It is best practice to not assume that arrays are filled with zeros when created.
ASSIGN	F77	Assigns a statement label to a variable. Note: This statement is a deleted feature in the Fortran standard, but remains supported in the Arm Fortran Compiler.
ASSOCIATE	F2003	Associates a name either with a variable or with the value of an expression, while in a block.
ASYNCHRONOUS	F77	Warns the compiler that incorrect results might occur for optimizations involving movement of code across wait statements, or statements that cause wait operations.
BACKSPACE	F77	Positions the file that is connected to the specified unit, to before the preceding record.

Table 5-20 Supported Fortran statements (continued)

Statement	Language standard	Brief description
BLOCK	F08	<p>Indicates where a BLOCK construct starts. The BLOCK construct defines an executable block of statements or constructs that can contain declarations. This allows you to declare variables closer to where they are used in your code.</p> <hr/> <p style="text-align: center;">Note</p> <ul style="list-style-type: none"> To retain the status and value of a local variable of a BLOCK construct after the block ends, use the SAVE attribute. SAVE-ed statements external to a block do not affect the local variables used internally in a block. Control can not be transferred into a block from outside the block, except when the return is from a procedure call. Transfers in or out of the block are permitted. <hr/> <p>Syntax</p> <pre><optional-name> BLOCK <optional-specification-part> ! One or more specification statements <statement-block> ! Zero or more statements or constructs END BLOCK <optional-name></pre> <p>The following specification statements are not permitted:</p> <ul style="list-style-type: none"> COMMON EQUIVALENCE IMPLICIT INTENT NAMelist OPTIONAL SUBROUTINE VALUE
BLOCK DATA	F77	Introduces several non-executable statements that initialize data values in COMMON tables.
BYTE	F77 ext	Establishes the data type of a variable by explicitly attaching the name of a variable to a 1-byte integer, overriding implied data typing.
CALL	F77	Transfers control to a subroutine.
CASE	F90	Begins a case-statement-block portion of a SELECT CASE statement.
CHARACTER	F90	<p>Establishes the data type of a variable by explicitly attaching the name of a variable to a character data type, overriding the implied data typing.</p> <p>Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.</p>
CLOSE	F77	Terminates the connection of the specified file to a unit.
COMMON	F77	<p>Defines global blocks of storage that are either sequential or non-sequential. Can be either static or dynamic form.</p> <p>Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.</p>

Table 5-20 Supported Fortran statements (continued)

Statement	Language standard	Brief description
COMPLEX	F90	Establishes the data type of a variable by explicitly attaching the name of a variable to a complex data type, overriding implied data typing.
CONTAINS	F90 F2003	Precedes a subprogram, a function or subroutine, and indicates the presence of the subroutine or function definition inside a main program, external subprogram, or module subprogram. In F2003, a CONTAINS statement can also appear in a derived type immediately before any type-bound procedure definitions.
CONTINUE	F77	Passes control to the next statement.
CYCLE	F90	Interrupts a DO construct execution and continues with the next iteration of the loop.
DATA	F77	Assigns initial values to variables before execution. Note: This statement amongst execution statements has been marked as obsolescent. This functionality is redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
DEALLOCATE	F90	Causes the memory that is allocated for each pointer-based variable or allocatable array that appears in the statement to be deallocated (freed). Also might be used to deallocate storage for deferred-shape arrays.
DECODE	F77 ext	Transfers data between variables or arrays in internal storage and translates that data from character form to internal form, according to format specifiers.
DIMENSION	F90	Defines the number of dimensions in an array and the number of elements in each dimension.
DO (Iterative)	F90	Introduces an iterative loop and specifies the loop control index and parameters. Note: Label form DO statements have been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
DO WHILE	F77	Introduces a logical DO loop and specifies the loop control expression.
DOUBLE COMPLEX	F77	Establishes the data type of a variable by explicitly attaching the name of a variable to a double complex data type. This overrides the implied data typing.
DOUBLE PRECISION	F90	Establishes the data type of a variable by explicitly attaching the name of a variable to a double precision data type, overriding implied data typing.
ELSE	F77	Begins an ELSE block of an IF block, and encloses a series of statements that are conditionally executed.
ELSE IF	F77	Begins an ELSE IF block of an IF block series, and encloses statements that are conditionally executed.
ELSE WHERE	F90	The portion of the WHERE ELSE WHERE construct that permits conditional masked assignments to the elements of an array, or to a scalar, zero-dimensional array.
ENCODE	F77 ext	Transfers data between variables or arrays in internal storage and translates that data from internal to character form, according to format specifiers.
END	F77	Terminates a segment of a Fortran program.
END ASSOCIATE	F2003	Terminates an ASSOCIATE block.

Table 5-20 Supported Fortran statements (continued)

Statement	Language standard	Brief description
END DO	F77	Terminates a DO or DO WHILE loop.
END FILE	F77	Writes an ENDFILE record to the files.
END IF	F77	Terminates an IF ELSE or ELSE IF block.
END MAP	F77 ext	Terminates a MAP declaration.
END SELECT	F90	Terminates a SELECT declaration.
END STRUCTURE	F77 ext	Terminates a STRUCTURE declaration.
END UNION	F77 ext	Terminates a UNION declaration.
END WHERE	F90	Terminates a WHERE ELSE WHERE construct.
ENTRY	F77	Allows a subroutine or function to have more than one entry point. Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
EQUIVALENCE	F77	Allows two or more named regions of data memory to share the same start address. Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
ERROR STOP	F2008	Stops the program execution and prevents any further execution of the program. ERROR STOP is similar to STOP, but ERROR STOP indicates that the program terminated in an error condition. Note: Also see STOP.
EXIT	F90	Interrupts a DO construct execution and continues with the next statement after the loop.
EXTERNAL	F77	Identifies a symbolic name as an external or dummy procedure which can then be used as an argument.
FINAL	F2003	Specifies a final subroutine inside a derived type.
FORALL	F95	Provides, as a statement or construct, a parallel mechanism to assign values to the elements of an array. Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
FORMAT	F77	Specifies format requirements for input or output.
FUNCTION	F77	Introduces a program unit; all the statements that follow apply to the function itself.
GENERIC	F2003	Specifies a generic type-bound procedure inside a derived type.
GOTO (Assigned)	F77	Transfers control so that the statement identified by the statement label is executed next. Note: This statement is a deleted feature in the Fortran standard, but remains supported in the Arm Fortran Compiler.

Table 5-20 Supported Fortran statements (continued)

Statement	Language standard	Brief description
GOTO (Computed)	F77	Transfers control to one of a list of labels, according to the value of an expression. Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
GOTO (Unconditional)	F77	Unconditionally transfers control to the statement with the label, which must be declared in the code of the program unit containing the GOTO statement, and also must be unique in that program unit.
IF (Arithmetic)	F77	Transfers control to one of three labeled statements, depending on the value of the arithmetic expression. Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
IF (Block)	F77	Consists of a series of statements that are conditionally executed.
IF (Logical)	F77	Executes or does not execute a statement based on the value of a logical expression.
IMPLICIT	F77	Redefines the implied data type of symbolic names from their initial letter, overriding implied data types.
IMPORT	F2003	Gives access to the named entities of the containing scope.
INCLUDE	F77 ext	Directs the compiler to start reading from another file.
INQUIRE	F77	Inquires about the current properties of a particular file or the current connections of a particular unit.
INTEGER	F77	Establishes the data type of a variable by explicitly attaching the name of a variable to an integer data type, overriding implied data types.
INTENT	F90	Specifies the intended use of a dummy argument, but can not be used in a specification statement of a main program.
INTERFACE	F90	Makes an implicit procedure an explicit procedure where the dummy parameters and procedure type are known to the calling module; Also overloads a procedure name.
INTRINSIC	F77	Identifies a symbolic name as an intrinsic function and allows it to be used as an actual argument.
LOGICAL	F77	Establishes the data type of a variable by explicitly attaching the name of a variable to a logical data type, overriding implied data types.
MAP	F77 ext	Designates each unique field or group of fields in a UNION statement.
MODULE	F90	Specifies the entry point for a Fortran 90, or Fortran 95, module program unit. A module defines a host environment of scope of the module, and might contain subprograms that are in the same scoping unit.
NAMelist	F90	Allows the definition of NAMelist groups for NAMelist-directed I/O.
NULLIFY	F90	Disassociates a pointer from its target.
OPEN	F77	Connects an existing file to a unit, creates and connects a file to a unit, creates a file that is pre-connected, or changes certain specifiers of a connection between a file and a unit.
OPTIONAL	F90	Specifies dummy arguments that can be omitted or that are optional.

Table 5-20 Supported Fortran statements (continued)

Statement	Language standard	Brief description
OPTIONS	F77 ext	Confirms or overrides certain compiler command-line options.
PARAMETER	F77	Gives a symbolic name to a constant.
PAUSE	F77	Stops program execution. Note: This statement is a deleted feature in the Fortran standard, but remains supported in the Arm Fortran Compiler.
POINTER	F90	Provides a means for declaring pointers.
PRINT	F77	Transfers data to the standard output device from the items that are specified in the output list and format specification.
PRIVATE	F90 F2003	Specifies that entities that are defined in a module are not accessible outside of the module. PRIVATE can also appear inside a derived type to disallow access to its data components outside the defining module. In F2003, to disallow access to type-bound procedures outside the defining module, a PRIVATE statement can appear after a CONTAINS statement, in a derived type.
PROCEDURE	F2003	Specifies a type-bound procedure, procedure pointer, module procedure, dummy procedure, intrinsic procedure, or an external procedure.
PROGRAM	F77	Specifies the entry point for a linked Fortran program.
PROTECTED	F2003	Protects a module variable against modification from outside the module in which it was declared.
PUBLIC	F90	Specifies that entities that are defined in a module are accessible outside of the module.
PURE	F95	Indicates that a function or subroutine has no side effects.
READ	F77	Transfers data from the standard input device to the items specified in the input and format specifications.
REAL	F90	Establishes the data type of a variable by explicitly attaching the name of a variable to a data type, overriding implied data types.
RECORD	F77 ext	A VAX Fortran extension, defines a user-defined aggregate data item.
RECURSIVE	F90	Indicates whether a function or subroutine can call itself recursively.
RETURN	F77	When used in a subroutine, causes a return to the statement following a CALL. When used in a function, returns to the relevant arithmetic expression. Note: This statement has been marked as obsolescent. Obsolescent statements are now redundant and might be removed from future standards. This statement remains supported in the Arm Fortran Compiler.
REWIND	F77	Positions the file at the start. The statement has no effect if the file is already positioned at the start, or if the file is connected but does not exist.
SAVE	F77	Retains the definition status of an entity after a RETURN or END statement in a subroutine or function that has been executed.
SELECT CASE	F90	Begins a CASE construct.

Table 5-20 Supported Fortran statements (continued)

Statement	Language standard	Brief description
SELECT TYPE	F2003	Provides the capability to execute alternative code depending on the dynamic type of a polymorphic entity, and to gain access to dynamic parts. The alternative code is selected using the TYPE IS statement for a specific dynamic type, or the CLASS IS statement for a specific type (and all its type extensions). Use the optional class default statement to specify all other dynamic types that do not match a specified TYPE IS or CLASS IS statement. Like the CASE construct, the code consists of a several blocks and, at most, one is selected for execution.
SEQUENCE	F90	A derived type qualifier that specifies the ordering of the storage that is associated with the derived type. This statement specifies storage for use with COMMON and EQUIVALENCE statements.
STOP	F77	Stops program execution and precludes any further execution of the program. Note: Also see ERROR STOP.
STRUCTURE	F77 ext	A VAX extension to FORTRAN 77 that defines an aggregate data type.
SUBROUTINE	F77	Introduces a subprogram unit.
TARGET	F90	Specifies that a data type can be the object of a pointer variable (for example, pointed to by a pointer variable). Types that do not have the TARGET attribute cannot be the target of a pointer variable.
THEN	F77	Part of an IF block statement, surrounds a series of statements that are conditionally executed.
TYPE	F90 F2003	Begins a derived type data specification or declares variables of a specified user-defined type. Use the optional EXTENDS statement with TYPE to indicate a type extension in F2003.
UNION	F77 ext	A multi-statement declaration defining a data area that can be shared intermittently during program execution by one or more fields or groups of fields.
USE	F90	Gives a program unit access to the public entities or to the named entities in the specified module.
VOLATILE	F77 ext	Inhibits all optimizations on the variables, arrays and common blocks that it identifies.
WAIT	F2003	Performs a wait operation for specified pending asynchronous data transfer operations.
WHERE	F90	Permits masked assignments to the elements of an array or to a scalar, zero-dimensional array.
WRITE	F77	Transfers data to the standard output device from the items that are specified in the output list and format specification.

*See [WG5 Fortran Standards](#)**Note**

The denoted language standards indicate the standard that they were introduced in, or the standard that they were last significantly changed.

Related information[WG5 Fortran Standards](#)

Chapter 6

Standards support

Describes the level of support in Arm Fortran Compiler for Fortran and OpenMP standards.

It contains the following sections:

- [6.1 Fortran 2003](#) on page 6-170.
- [6.2 Fortran 2008](#) on page 6-173.
- [6.3 OpenMP 4.0](#) on page 6-176.
- [6.4 OpenMP 4.5](#) on page 6-177.

6.1 Fortran 2003

Details the support status with the Fortran 2003 standard.

Table 6-1 Fortran 2003 support

Fortran 2003 Feature	Support Status
ISO TR 15580 IEEE Arithmetic	Yes
ISO TR 15581 Allocatable Enhancements	
Dummy arrays	Yes
Function results	Yes
Structure components	Yes
Data enhancements and object orientation	
Parameterized derived types	Yes
Procedure pointers	Yes
Finalization	Yes
Procedures that are bound by name to a type	Yes
The PASS attribute	Yes
Procedures that are bound to a type as operators	Yes
Type extension	Yes
Overriding a type-bound procedure	Yes
Enumerations	Yes
ASSOCIATE construct	Yes
Polymorphic entities	Yes
SELECT TYPE construct	Yes
Deferred bindings and abstract types	Yes
Allocatable scalars	Yes
Allocatable character length	Yes
Miscellaneous enhancements	Yes
Structure constructor changes	Yes
Generic procedure interfaces with the same name as a type	Yes
The allocate statement	Yes
Source specifier	Yes
Errmsg specifier	Yes
Assignment to an allocatable array	Yes
Transferring an allocation	Yes
More control of access from a module	Yes
Renaming operators on the USE statement	Yes
Pointer assignment	Yes

Table 6-1 Fortran 2003 support (continued)

Fortran 2003 Feature	Support Status
Pointer INTENT	Yes
The VOLATILE attribute	Yes One or more issues are observed with this feature.
The IMPORT statement	Yes
Intrinsic modules	Yes
Access to the computing environment	Yes
Support for international character sets	Partial Only <code>selected_char_kind</code> is supported.
Lengths of names and statements	
names = 63	Yes
statements = 256	Yes
Binary, octal and hex constants	Yes
Array constructor syntax	Yes
Specification and initialization expressions	Yes A few intrinsics which are not commonly used are not supported.
Complex constants	Yes
Changes to intrinsic functions	Yes
Controlling IEEE underflow	Yes
Another IEEE class value	Yes
I/O enhancements	Yes
Derived type I/O	Yes One or more issues are observed with this feature.
Asynchronous I/O	Yes One or more issues are observed with this feature.
FLUSH statement	Yes
IOMSG= specifier	Yes
Stream access input/output	Yes
ROUND= specifier	Yes Not supported for write.
DECIMAL= specifier	Yes
SIGN= specifier	Yes <code>processor_defined</code> does not work for open.
Kind type parameters of integer specifiers	Yes

Table 6-1 Fortran 2003 support (continued)

Fortran 2003 Feature	Support Status
Recursive input/output	Yes
Intrinsic function for newline character	Yes
Input and output of IEEE exceptional values	Yes Read does not work for NaN(s).
Comma after a P edit descriptor	Yes
Interoperability with	
Interoperability of intrinsic types	Yes
Interoperability with C pointers	Yes
Interoperability of derived types	Yes
Interoperability of variables	Yes
Interoperability of procedures	Yes
Interoperability of global data	Yes

Note

For more information about the features that are listed in the table above, see [N1648 – ISO/IEC JTC1/SC22/WG5: The new features of Fortran 2003](#).

6.2 Fortran 2008

Details the support status with the Fortran 2008 standard.

Table 6-2 Fortran 2008 support

Fortran 2008 feature	Support status
Submodules	Yes
Coarrays	No
Performance enhancements	
do concurrent	Partial The <code>do concurrent</code> syntax is accepted. The code that is generated is serial.
Contiguous attribute	Yes
Data Declaration	
Maximum rank + corank = 15	No
Long integers	Yes
Allocatable components of recursive type	No
Implied-shape array	No
Pointer initialization	No
Data statement restrictions lifted	No
Kind of a forall index	No
Type statement for intrinsic types	No
Declaring type-bound procedures	Yes Supports declaring multiple type-bound procedures in a single procedure statement.
Value attribute is permitted for any nonallocatable nonpointer noncoarray	No
In a pure procedure the intent of an argument need not be specified if it has the value attribute	Yes
Accessing data objects	
Simply contiguous arrays rank remapping to rank>1 target	Yes
Omitting an ALLOCATABLE component in a structure constructor	No
Multiple allocations with SOURCE=	No
Copying the properties of an object in an ALLOCATE statement	Yes
MOLD= specifier for ALLOCATE	Yes
Copying bounds of source array in ALLOCATE	Yes
Polymorphic assignment	No
Accessing real and imaginary parts	Partial Not supported for complex arrays.

Table 6-2 Fortran 2008 support (continued)

Fortran 2008 feature	Support status
Pointer function reference is a variable	No
Elemental dummy argument restrictions lifted	Yes
Input/Output	
Finding a unit when opening a file	Yes
g0 edit descriptor	No
Unlimited format item	Yes
Recursive I/O	Yes
Execution control	
The BLOCK construct	Yes
Exit statement	No
Stop code	Yes
ERROR STOP	Yes
Intrinsic procedures for bit processing	
Bit sequence comparison	No
Combined shifting	No
Counting bits	Yes
Masking bits	No
Shifting bits	No
Merging bits	No
Bit transformational functions	No
Intrinsic procedures and modules	
Storage size	Yes
Optional argument RADIX added to SELECTED REAL	No
Extensions to trigonometric and hyperbolic intrinsics	Partial Complex types are not accepted for acosh, asinh and atanh. Also, atan2 cannot be accessed through atan.
Bessel functions	Yes
Error and gamma functions	Yes
Euclidean vector norms	Yes The current implementation experiences overflow for arguments containing elements whose square is at the boundary value for double-precision floating-point numbers. There is no such overflow for single-precision arguments.

Table 6-2 Fortran 2008 support (continued)

Fortran 2008 feature	Support status
Parity	No
Execute command line	No
Optional back argument added to maxloc and minloc	Yes
Find location in an array	Yes
String comparison	Yes
Constants	Yes
COMPILER_VERSION	Yes
COMPILER_OPTIONS	Yes
Function for C sizeof	Yes
Added optional argument for IEEE_SELECTED_REAL_KIND	No
Programs and procedures	
Save attribute for module and submodule data	Partial One or more issues are observed with this feature.
Empty contains section	Partial Not supported for procedures.
Form of end statement for internal and module procedures	Yes
Internal procedure as an actual argument	Yes
Null pointer or unallocated allocatable as absent dummy arg.	Partial Not supported for null pointer.
Non pointer actual for pointer dummy argument	Yes
Generic resolution by procedureness	No
Generic resolution by pointer vs. allocatable	Yes
Impure elemental procedures	Yes
Entry statement becomes obsolescent	Yes
Source form	
Semicolon at line start	Yes

Note

For more information about the features that are listed in the table above, see *N1891 – ISO/IEC JTC1/SC22/WG5: The new features of Fortran 2008*.

6.3 OpenMP 4.0

Details the support status with the OpenMP 4.0 standard.

Table 6-3 OpenMP 4.0 support

OpenMP 4.0 Feature	Support
C/C++ Array Sections	N/A
Thread affinity policies	Yes
“simd” construct	Partial Note: No clauses are supported. <code>!\$omp simd</code> can be used to force a loop to be vectorized.
“declare simd” construct	No
Device constructs	No
Task dependencies	No
“taskgroup” construct	Yes
User defined reductions	No
Atomic capture swap	Yes
Atomic seq_cst	No
Cancellation	Yes
OMP_DISPLAY_ENV	Yes

6.4 OpenMP 4.5

Details the support status with the OpenMP 4.5 standard.

Table 6-4 OpenMP 4.5 support

OpenMP 4.5 Feature	Support
doacross loop nests with ordered	No
“linear” clause on loop construct	No
“simdlen” clause on simd construct	No
Task priorities	No
“taskloop” construct	Yes
Extensions to device support	No
“if” clause for combined constructs	Yes
“hint” clause for critical construct	No
“source” and “sink” dependence types	No
C++ reference types in data sharing attribute clauses	N/A
Reductions on C/C++ array sections	N/A
“ref”, “val”, “uval” modifiers for linear clause	No
Thread affinity query functions	Yes
Hints for lock API	Yes

Chapter 7

Troubleshoot

Describes how to diagnose problems when compiling applications using Arm Fortran Compiler.

It contains the following sections:

- [7.1 Application segfaults at -Ofast optimization level on page 7-179.](#)
- [7.2 Compiling with the -fpic option fails when using GCC compilers on page 7-180.](#)
- [7.3 Error messages when installing Arm® Compiler for Linux on page 7-181.](#)

7.1 Application segfaults at -Ofast optimization level

A Fortran program runs correctly when the binary is built with `armflang` at `-O3` level, but encounters a runtime crash or segfault with `-Ofast` optimization level.

Condition

The runtime segfault only occurs when `-Ofast` is used to compile the code. The segfault disappears when you add the `-fno-stack-arrays` option at the compilation with `armflang`.

The `-fstack-arrays` option is enabled by default at `-Ofast`

When the `-fstack-arrays` option is enabled, either on its own or enabled with `-Ofast` by default, the compiler allocates arrays for all sizes using the local stack for local and temporary arrays. This helps to improve performance, because it avoids slower heap operations with `malloc()` and `free()`. However, applications that use large arrays might reach the Linux stack-size limit at runtime and produce program segfaults. On typical Linux systems, a default stack-size limit is set, such as 8192 kilobytes. You can adjust this default stack-size limit to a suitable value.

Solution

Use `-Ofast -fno-stack-arrays` instead. This disables automatic arrays on the local stack, and keeps all other `-Ofast` optimizations. Alternatively, to set the stack so that it is larger than the default size, call `ulimit -s unlimited` before running the program.

If you continue to experience problems, [Contact Arm Support](#).

7.2 Compiling with the `-fpic` option fails when using GCC compilers

Describes the difference between the `-fpic` and `-fPIC` options when compiling for Arm with GCC and Arm Compiler for Linux.

Condition

Failure can occur at the linking stage when building Position-Independent Code (PIC) on AArch64 using the lower-case `-fpic` compiler option with GCC compilers (gfortran, gcc, g++), in preference to using the upper-case `-fPIC` option.

Note

- This issue does not occur when using the `-fpic` option with Arm Compiler for Linux (armclang/armclang++), and it also does not occur on x86_64 because `-fpic` operates the same as `-fPIC`.
- PIC is code which is suitable for shared libraries.

Cause

Using the `-fpic` compiler option with GCC compilers on AArch64 causes the compiler to generate one less instruction per address computation in the code, and can provide code size and performance benefits. However, it also sets a limit of 32k for the Global Offset Table (GOT), and the build can fail at the executable linking stage because the GOT overflows.

Note

When building PIC with Arm Compiler for Linux on AArch64, or building PIC on x86_64, `-fpic` does not set a limit for the GOT, and this issue does not occur.

Solution

Consider using the `-fPIC` compiler option with GCC compilers on AArch64, because it ensures that the size of the GOT for a dynamically linked executable will be large enough to allow the entries to be resolved by the dynamic loader.

7.3 Error messages when installing Arm® Compiler for Linux

If you experience a problem when installing Arm Compiler for Linux, consider the following points.

- To perform a system-wide install, ensure that you have the correct permissions. If you do not have the correct permissions, the following errors are returned:

— Systems using RPM Package Manager (RPM):

```
error: can't create transaction lock on /var/lib/rpm/.rpm.lock (Permission denied)
```

— Debian systems using dpkg:

```
dpkg: error: requested operation requires superuser privilege
```

- If you install using the `--install-to <directory>` option, ensure that the system you are installing on has the required rpm or dpkg binaries installed. If it does not, the following errors are returned:

— Systems using RPM Package Manager (RPM):

```
Cannot find 'rpm' on your PATH. Unable to extract .rpm files.
```

— Debian systems using dpkg:

```
Cannot find 'dpkg' on your PATH. Unable to extract .deb files.
```