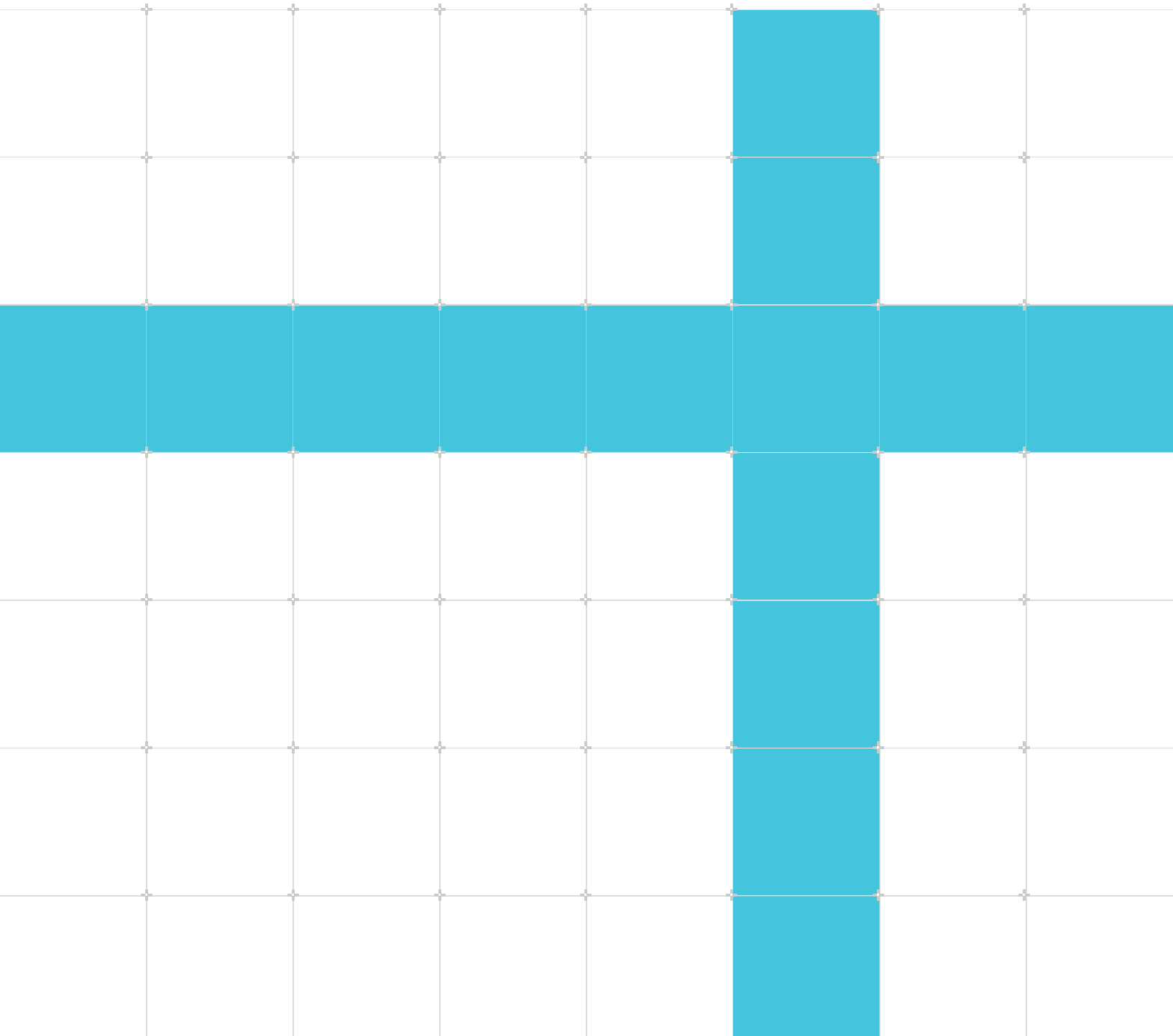
Arm Toolchain for Linux Reference Guide

Version 20.1

Issue 00
110477_201_00_en



Arm Toolchain for Linux Reference Guide

This document (110477_201_00_en) was issued on 2025-04-30. There might be a later issue at https://developer.arm.com/documentation/110477

The product version is 20.1.

See also: Proprietary notice | Product and document information | Useful resources

Start reading

If you prefer, you can skip to the start of the content.

Intended audience

Software developers

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Contents

1. Welcome to the Arm Toolchain For Linux

The Arm Toolchain for Linux provides a complete compiling environment for natively developing and tuning your server and HPC applications on Arm-based platforms.

1.1 Installation

This document describes how to download and install the Arm Toolchain for Linux.

1.1.1 Pre-installation step

The first step is to configure your Linux package manager to be able to fetch packages from Arm. This step only needs to be performed once.

From the options below, select the packages repository matching with your installed Linux distribution.

Ubuntu 22.04

```
$ curl "https://developer.arm.com/packages/arm-toolchains:ubuntu-22/jammy/
Release.key" | sudo gpg --dearmor -o /usr/share/keyrings/obs-oss-arm-com.gpg

$ echo "deb [signed-by=/usr/share/keyrings/obs-oss-arm-com.gpg] https://
developer.arm.com/packages/arm-toolchains:ubuntu-22/jammy/ ./" | sudo tee /etc/apt/
sources.list.d/obs-oss-arm-com.list

$ sudo apt update
```

Ubuntu 24.04

```
$ curl "https://developer.arm.com/packages/arm-toolchains:ubuntu-24/noble/
Release.key" | sudo gpg --dearmor -o /usr/share/keyrings/obs-oss-arm-com.gpg

$ echo "deb [signed-by=/usr/share/keyrings/obs-oss-arm-com.gpg] https://
developer.arm.com/packages/arm-toolchains:ubuntu-24/noble/ ./" | sudo tee /etc/apt/
sources.list.d/obs-oss-arm-com.list

$ sudo apt update
```

Red Hat Enterprise Linux 8

```
$ sudo dnf install 'dnf-command(config-manager)'

$ sudo dnf config-manager -y --add-repo https://developer.arm.com/packages/arm-
toolchains:rhel-8/el8/arm-toolchains:rhel-8.repo
```

Red Hat Enterprise Linux 9

```
$ sudo dnf install 'dnf-command(config-manager)'
```

```
$ sudo dnf config-manager -y --add-repo https://developer.arm.com/packages/arm-toolchains:rhel-9/el9/arm-toolchains:rhel-9.repo
```

Amazon Linux 2023

```
$ sudo dnf install 'dnf-command(config-manager)'

$ sudo dnf config-manager -y --add-repo https://developer.arm.com/packages/arm-toolchains:amzn-2023/al2023/arm-toolchains:amzn-2023.repo
```

SUSE Linux Enterprise Server 15

```
$ sudo zypper ar -f https://developer.arm.com/packages/arm-toolchains:sles-15/sl15/arm-toolchains:sles-15.repo
```

1.1.2 Installation step

Please select the command below appropriate for your Linux distribution. This will install Arm Toolchain For Linux, along with Arm Performance Libraries, which is a required dependency.

Ubuntu systems

```
$ sudo apt install arm-toolchain-for-linux
```

Red Hat Enterprise Linux and Amazon Linux systems

```
$ sudo dnf install arm-toolchain-for-linux
```

SUSE Linux Enterprise Server systems

```
$ sudo zypper install arm-toolchain-for-linux
```

1.2 Getting Started

This document describes how to get started with the already installed Arm Toolchain for Linux. Here you will find out how to use it to compile a source code into an executable binary.

1.2.1 Looking at the toolchain

The default installation site of Arm Toolchain For Linux is the `/opt/arm/arm-toolchain-for-linux` directory. It contains a complete set of LLVM tooling, header files, compiler libraries and runtime libraries (including the OpenMP runtime). The main tools are as follows:

- `armclang` - the C compiler
- `armclang++` - the C++ compiler
- `armflang` - the Fortran compiler

Note that the LLVM equivalents of these commands (`clang`, `clang++`, `flang`) are also available and functionally identical.

1.2.2 Configure and load environment modules

After installation, you should be able to invoke any of those commands with their absolute paths, e.g.:

```
$ /opt/arm/arm-toolchain-for-linux/bin/armclang -print-resource-dir
/opt/arm/arm-toolchain-for-linux/lib/clang/<version>
```

Although fully operable, this is not the most convenient way of using the toolchain. Therefore, we recommend using the environment modules. These can be installed on most of the existing Linux distributions, as presented below.

Ubuntu systems

```
$ sudo apt install environment-modules
```

Red Hat Enterprise Linux and Amazon Linux systems

```
$ sudo dnf install environment-modules
```

SUSE Linux Enterprise Server systems

```
$ sudo zypper install environment-modules
```

After installing environment modules, you need to execute a profile script which matches with your default shell:

BASH

```
$ source /etc/profile.d/modules.sh
```

csh or tcsh

```
$ source /etc/profile.d/modules.csh
```

Use the `module use` command to update your `MODULEPATH` environment variable to include the path to the Arm Toolchain For Linux module files directory:

```
$ module use /opt/arm/modulefiles
```

Alternatively, you can set or amend your `MODULEPATH` environment variable manually. Use the `module avail` command to examine the list of available modules.

To load the module for Arm Toolchain for Linux, type:

```
$ module load atfl
```

This should load the default version of Arm Toolchain for Linux. Alternatively, if multiple versions are available, you can load the desired version by specifying `atfl/<version>`.

From now on, the toolchain commands should be accessible from the command line:

```
$ which armclang
/opt/arm/arm-toolchain-for-linux/bin/armclang

$ which armclang++
/opt/arm/arm-toolchain-for-linux/bin/armclang++

$ which armflang
/opt/arm/arm-toolchain-for-linux/bin/armflang
```

1.2.3 Using the compiler

Example 1: Compile and run an example C program

Consider a simple program stored in a `.c` file, for example: `hello.c`:

```
#include <stdio.h>

int main()
{
  printf("Hello, World!");
  return 0;
}
```

To generate an executable binary, compile your example C program with the `armclang` compiler. Specify the input file, `hello.c`, and the binary name (using `-o`), `hello`:

```
$ armclang -o hello hello.c
```

Run the generated binary `hello`:

```
$ ./hello
Hello, World!
```

Example 2: Compile and run an example Fortran program

Consider a simple program stored in a `.f90` file, for example: `hello.f90`:

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

To generate an executable binary, compile your example Fortran program with the `armflang` compiler. Specify the input file, `hello.f90`, and the binary name (using `-o`), `hello`:

```
$ armflang -o hello hello.f90
```

Run the generated binary `hello`:

```
$ ./hello
 hello world
```

1.2.4 Compile and link C/C++ programs

To generate an executable binary, compile your source file (for example, `source.c`) with the `armclang` command:

```
$ armclang source.c
```

A binary with the filename `a.out` is the output.

Optionally, use the `-o` option to set the binary filename (for example, `binary`):

```
$ armclang -o binary source.c
```

You can 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 the source files `source1.c` and `source2.c`, use:

```
$ armclang -o binary source1.c source2.c
```

To compile each of your source files individually into an object (`.o`) file, specify the compile-only option, `-c`, and then pass the resulting object files into another invocation of `armclang` to link them into an executable binary:

```
$ armclang -c source1.c
$ armclang -c source2.c
$ armclang -o binary source1.o source2.o
```

1.2.4.1 Note

For the C/C++ compiler's command line arguments reference visit this page: Clang command line argument reference.

1.2.5 Compile and link Fortran programs

To generate an executable binary, compile your source file (for example, `source.f90`) with the `armflang` command:

```
$ armflang source.f90
```

A binary with the filename `a.out` is the output.

You can 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 the source files `source1.f90` and `source2.f90`, use:

```
$ armflang -o binary source1.f90 source2.f90
```

To compile each of your source files individually into an object (`.o`) file, specify the compile-only option, `-c`, and then pass the resulting object files into another invocation of `armflang` to link them into an executable binary:

```
$ armflang -c source1.f90
$ armflang -c source2.f90
$ armflang -o binary source1.o source2.o
```

When mixing both C/C++ and Fortran codes in a single application, it is important to to make sure that the Fortran runtime library is always linked in. This can be ensured by using the `armflang` command for linking:

```
$ armflang -c source1.f90
$ armclang -c source2.c
```

```
$ armflang -o binary source1.o source2.o
```

1.2.5.1 Note

For the Fortran compiler's command line arguments reference visit this page: Flang command line argument reference.

1.2.6 Increase the optimization level

To control the optimization level, specify the `-O<level>` option on your compile line, and replace `<level>` with one of `0`, `1`, `2` or `3`. The `-O0` option is the lowest, and the default, optimization level. `-O3` is the highest optimization level. Arm compilers performs auto-vectorization at level `-O2` and above.

For example, to compile the `source.c` file into a binary called `binary`, and use the `-O3` optimization level, use:

```
$ armclang -O3 -o binary source.c
```

To compile the `source.f90` file into a binary called binary, and use the `-O3` optimization level, use:

```
$ armflang -O3 -o binary source.f90
```

1.2.6.1 Note

Similarly to other compilers, the Arm Toolchain for Linux C and C++ compilers can also be supplied with the `-Ofast` option. This will however result in displaying the following deprecation warning:

```
warning: argument '-Ofast' is deprecated; use '-O3 -ffast-math' for the same
 behavior, or '-O3' to enable only conforming optimizations [-Wdeprecated-ofast]
```

As the warning message states, the effect of applying the `-Ofast` option when compiling the C/C++ programs can be achieved by using the `-O3 -ffast-math` options instead.

In case of Fortran, the use of the `-Ofast` option triggers the following deprecation warning:

```
warning: argument '-Ofast' is deprecated; use '-O3 -ffast-math -fstack-arrays' for
 the same behavior, or '-O3 -fstack-arrays' to enable only conforming optimizations
 [-Wdeprecated-ofast]
```

As the warning message states, the effect of applying the `-Ofast` option when compiling Fortram programs can be achieved by using the `-O3 -ffast-math -fstack-arrays` options instead.

1.2.7 Compile and optimize using CPU auto-detection

If you tell the compiler what target CPU your application will run on, it can make target-specific optimization decisions. Target-specific optimization decisions help ensure your application runs as efficiently as possible. To tell the compiler to make target-specific compilation decisions, use the `-mcpu=<target>` option and replace `<target>` with your target processor (from a supported list of targets).

In addition, the `-mcpu` option also supports the `native` argument. `-mcpu=native` enables the compiler to auto-detect the architecture and processor type of the CPU that you are running the compiler on.

For example, to auto-detect the target CPU and optimize your C application for this target, use:

```
$ armclang -O3 -mcpu=native -o binary source.c
```

To auto-detect the target CPU and optimize your Fortran application for this target, use:

```
$ armflang -O3 -mcpu=native -o binary source.f90
```

The `-mcpu` option supports a range of Armv8-A-based Systems-on-Chips (SoCs). When `-mcpu` is not specified, by default, `-mcpu=generic` is set, which generates portable output suitable for any Armv8-A-based target.

Note

- The optimizations that are performed from setting the `-mcpu` option (also

  known as hardware, or CPU, tuning) are independent of the optimizations that are performed from setting the `-O<level>` option.
- If you run the compiler on one target, but will run the application you are

  compiling on a different target, do not use `-mcpu=native`. Instead, use `-mcpu=<target>` where `<target>` is the target processor that you will run the application on.

1.2.8 Standards support

For information on C, C++, and Fortran language support in ATfL, see the Standards Support page. For details on OpenMP support in ATfL, refer to the OpenMP Support page.

1.2.9 Fortran Recommendations

Who should use Arm Toolchain For Linux

- Code with modern Fortran features (except coarrays/teams/collectives) will

work with ATfL.
- Applications that are standard compliant will work with ATfL.
- Applications like CP2K can be compiled with ATfL.
- Applications requiring quadmath support can be compiled with ATfL.

Who should not use Arm Toolchain For Linux
- Performance is not guaranteed. For users seeking highest performance ATfL is not recommended.
- OpenMP support is experimental.
- Code containing non-standard features/intrinsics might not work as expected.
- CMake versions older than 3.28 are not supported.

1.3 Using the Arm Performance Libraries

You can get greater performance from your code if you enable linking to the optimized math libraries at compilation time.

1.3.1 Linking to Arm Performance Libraries

To enable you to get the best performance on Arm-based systems, Arm recommends linking to Arm Performance Libraries. Arm Performance Libraries provide optimized standard core math libraries for high-performance computing applications on Arm processors. Through a C interface, the following types of routines are available:

- BLAS: Basic Linear Algebra Subprograms (including XBLAS, the extended precision BLAS).
- LAPACK: A comprehensive package of higher level linear algebra routines.
- FFT functions: A set of Fast Fourier Transform routines for real and complex data using the FFTW interface.
- Sparse linear algebra.
- libastring: A subset of libc, which is a set of optimized string functions.

The easiest way to make the Arm Performance Libraries visible to the compiler is to load the `arm-performance-libraries` environment module. After the `atfl` environment module has been loaded, the arm-performance-libraries module should become available for loading:

```
$ module load arm-performance-libraries
```

This should set the `ARMPL_`-prefixed environment variables and two other critical environment variables: `LD_LIBRARY_PATH` and `PKG_CONFIG_PATH`. The recommended way of selecting command line flags for using a specific variant of Arm Performance Libraries is through invoking the `pkg-config` command. Note that the Arm Performance Libraries export several `pkg-config` modules, you should be picking the one that you actually need, particularly when you plan to use OpenMP (notice the `seq` vs. `omp` suffix):

```
$ pkg-config --list-all | grep armpl
armpl                           ArmPL - Arm Performance Libraries
armpl-Fortran-dynamic-ilp64-omp ArmPL - Arm Performance Libraries
armpl-Fortran-dynamic-ilp64-seq ArmPL - Arm Performance Libraries
armpl-Fortran-dynamic-lp64-omp  ArmPL - Arm Performance Libraries
armpl-Fortran-dynamic-lp64-seq  ArmPL - Arm Performance Libraries
armpl-Fortran-static-ilp64-omp  ArmPL - Arm Performance Libraries
armpl-Fortran-static-ilp64-seq  ArmPL - Arm Performance Libraries
armpl-Fortran-static-lp64-omp   ArmPL - Arm Performance Libraries
armpl-Fortran-static-lp64-seq   ArmPL - Arm Performance Libraries
armpl-dynamic-ilp64-omp         ArmPL - Arm Performance Libraries
armpl-dynamic-ilp64-seq         ArmPL - Arm Performance Libraries
armpl-dynamic-lp64-omp          ArmPL - Arm Performance Libraries
armpl-dynamic-lp64-seq          ArmPL - Arm Performance Libraries
armpl-static-ilp64-omp          ArmPL - Arm Performance Libraries
armpl-static-ilp64-seq          ArmPL - Arm Performance Libraries
armpl-static-lp64-omp           ArmPL - Arm Performance Libraries
armpl-static-lp64-seq           ArmPL - Arm Performance Libraries
```

1.3.1.1 C/C++ examples

To link to the OpenMP multi-threaded Arm Performance Libraries with a 64-bit integer interface, and include compiler and library optimizations for an Neoverse N1-based system, use:

```
$ armclang -fopenmp -o binary code_with_math_routines.c -mcpu=neoverse-n1 `pkg-
config armpl-dynamic-ilp64-omp --cflags --libs`
```

To link to the OpenMP multi-threaded Arm Performance Libraries with a 32-bit integer interface, and build portable output that is suitable for any Armv8-A-based system, use:

```
$ armclang -fopenmp -o binary code_with_math_routines.c -mcpu=generic `pkg-config
 armpl-dynamic-lp64-omp --cflags --libs`
```

To link to the serial implementation of Arm Performance Libraries with a 32-bit integer interface, and include compiler and library optimizations for an Neoverse V2-based system, use:

```
$ armclang -o binary code_with_math_routines.c -mcpu=neoverse-v2 `pkg-config armpl-
dynamic-lp64-seq --cflags --libs`
```

1.3.1.2 Fortran examples

To link to the OpenMP multi-threaded Arm Performance Libraries with a 64-bit integer interface, and include compiler and library optimizations for an Neoverse N2-based system, use:

```
$ armflang -fopenmp -o binary code_with_math_routines.f90 -mcpu=neoverse-n2 `pkg-
config armpl-dynamic-ilp64-omp --cflags --libs`
```

To link to the OpenMP multi-threaded Arm Performance Libraries with a 32-bit integer interface, and build portable output that is suitable for any Armv8-A-based system, use:

```
$ armflang -fopenmp -o binary code_with_math_routines.f90 -mcpu=generic `pkg-config
 armpl-dynamic-lp64-omp --cflags --libs`
```

To link to the serial implementation of Arm Performance Libraries with a 32-bit integer interface, and include compiler and library optimizations for an Neoverse V1-based system, use:

```
$ armflang -o binary code_with_math_routines.f90 -mcpu=neoverse-v1 `pkg-config
 armpl-dynamic-lp64-seq --cflags --libs`
```

1.3.1.3 More information

For more information please visit this page: Get started with Arm Performance Libraries (stand-alone Linux version) Version 24.10.

To learn more about integrating Arm Performance Libraries with Arm Toolchain For Linux please visit Using Arm Performance Libraries (ArmPL) with ATfL.

1.3.1.4 Note

The Arm Performance Libraries suite is also the provider of the vectorized math routines library (libamath). This is a subset of the libm functions, which makes it possible to vectorize the loops containing calls to those functions. The Arm Toolchain for Linux default configuration instructs the C/C++ and Fortran compilers to make use of this library during vectorization automatically, no further command line options are needed. This can be disabled by specifying the `-fveclib=none` option.

1.4 ACfL to ATfL Porting Guide

This document outlines the key differences between using the Arm Toolchain for Linux (ATfL) and the Arm Compiler for Linux (ACfL). It also lists command-line options and macros that can be used with ATfL as alternatives to those used in ACfL.

Both toolchains provide frontends for compiling C, C++, and Fortran code, and the names of these frontends are consistent across both: `armclang` for C, `armclang++` for C++, and `armflang` for Fortran.

The quadmath is supported by ATfL, while it was not supported by ACfL.

1.4.1 Reference Version

The table below lists the version numbers of the toolchains for which this porting guide is applicable.

Compiler	Version
Arm Compiler for Linux (ACfL)	24.10
Arm Toolchain for Linux (ATfL)	20.1

1.4.2 ArmPL integration

In ACfL, Arm Performance Libraries (ArmPL) are bundled with the compiler. To simplify usage, the `-armpl` flag is provided to automatically include the necessary headers and libraries for BLAS, LAPACK, FFT, and other numerical routines.

In contrast, ATfL does not include ArmPL directly, but it depends on the ArmPL package, which is installed alongside the toolchain. To use ArmPL with ATfL, the users must manually specify the locations of the headers and libraries using `pkg-config`. Detailed instructions are available in the Getting Started guide.

1.4.2.1 Note

Vector math functions from libamath are accessible without manually specifying the libraries.

1.4.3 C/C++ Frontend

The C/C++ frontend in ATfL is identical to the one used in ACfL. Both are built on the Clang frontend from the upstream LLVM project (`llvm-project/clang`).

1.4.4 Fortran Frontend

The Fortran frontend in ATfL is based on the new LLVM Flang project (`llvm-project/flang`). This frontend is a modern, from-scratch implementation. In contrast, ACfL used the older Classic Flang frontend, available at https://github.com/flang-compiler/flang. Because ATfL introduces a new Fortran frontend, this document will primarily focus on the differences in Fortran support between the two toolchains.

1.4.4.1 Major difference

The table below lists the major difference between the two toolchains.

	Compatibility
Binary	No
Module file	No
Array descriptor	No

1.4.4.2 Difference in Fortran features

The table below lists the difference in Fortran features between the two toolchains.

Feature	ACfL	ATfL
Base Fortran standard	Fortran 2008	Fortran 2018
Base OpenMP Specification	OpenMP 4.0	OpenMP 2.5 Note: OpenMP support is experimental
Parameterized Derived Type	Supported	PDT Kind - Supported PDT Length - Not supported
Preprocessor	Use `-cpp` to switch ON	Always ON Use `-cpp` to switch ON processing of predefined macros and macros from the command line
Directives	`ivdep`, `prefetch`, `unroll`, `nounroll`, `vector always`, `vector vectorlength`, `novector`	`vector always`
Line length	2100	Unlimited
Recursive functions	Use `-frecursive`	Default
Main function	Is a library linked at link-time	Is generated in the object file containing the program statement

1.4.4.3 Difference in command line flags

The following table summarises some of the most commonly used compiler flags in ACfL and gives their equivalent in ATfL:

ACfL	ATfL	Description

`-module <path>`	`-J <path>`
	
	`-I <path>`
		Specifies a directory to place module files
		
		Specifies a directory to search for module files

`-Mallocatable=03`	`-frealloc-lhs`	Use Fortran 2003 standard semantics for assignments to allocatables
`-Mallocatable=95`	`-fno-realloc-lhs`	Use pre-Fortran 2003 standard semantics for assignments to allocatables
`-r8`	`-fdefault-real-8`	Sets the default `KIND` for `REAL` and `COMPLEX` declarations, constants, functions, and intrinsics
`-i8`	`-fdefault-integer-8`	Set the default `KIND` for `INTEGER` and `LOGICAL` to 64bit (i.e., `KIND = 8`)

1.4.4.4 Pre-defined macros

`armflang` has the following compiler and machine specific predefined processor macros in the two toolchains:

ACfL	ATfL	Value	Description
`__FLANG`	`__flang__`	1	Selection of compiler dependent source at compile time
`__arch64`	N/A	1	Selection of architecture dependent source at compile time
`__aarch64__`	N/A	1	Selection of architecture dependent source at compile time
`__ARM_ARCH`	N/A	8	Selection of architecture dependent source at compile time
`__ARM_ARCH__`	N/A	8	Selection of architecture dependent source at compile time
`__armflang_major__`	`__flang_major__`	24/20	Underlying LLVM version details
`__armflang_minor__`	`__flang_minor__`	10/1	Underlying LLVM version details
`__armflang_version__`	`__flang_patchlevel__`	24.10.1/0	Underlying LLVM version details
`__linux__`	`__linux__`	1	Targeted Operating System
`__linux`	`__linux__`	1	Targeted Operating System
`linux`	`__linux__`	1	Targeted Operating System

2. Standards Support

This section describes the Standards Support for ATfL.

2.1 C Standard Support

Refer to the Clang C status page for details on C standard support.

2.2 C++ Standard Support

Refer to the Clang C++ status page for details on C++ standard support.

2.3 Fortran Standards Support

This section summarizes Fortran standards support in Flang. The information is only provided as a guideline. The TODOs/Not Yet Implemented messages emitted by the compiler for unimplemented features should be treated as authoritative.

The standards support information is provided as a table with three columns that are self explanatory. The Status column uses the letters P, Y, N for the implementation status:

- P : When the implementation is incomplete for a few cases
- Y : When the implementation is complete
- N : When the implementation is absent

Note: No distinction is made between the support in the Parser/Semantics and MLIR or Lowering support.

2.3.1 Fortran 2023

See this document for a brief discussion about the new features in Fortran 2023. The following table summarizes the status of all important Fortran 2023 features.

Feature	Status	Comments
Allow longer statement lines and overall statement length	Y	
Automatic allocation of lengths of character variables	N	
The specifiers typeof and classof	N	
Conditional expressions and arguments	N	
More use of boz constants	P	All usages other than enum are supported

Feature	Status	Comments
Intrinsics for extracting tokens from a string	N	
Intrinsics for Trig functions that work in degrees	N	
Intrinsics for Trig functions that work in half revolutions	N	
Changes to system_clock	N	
Changes for conformance with the new IEEE standard	Y	
Additional named constants to specify kinds	Y	
Extensions for c_f_pointer intrinsic	N	
Procedures for converting between fortran and c strings	N	
The at edit descriptor	N	
Control over leading zeros in output of real values	N	
Extensions for Namelist	N	
Allow an object of a type with a coarray ultimate component to be an array or allocatable	N	
Put with Notify	N	
Error conditions in collectives	N	
Simple procedures	N	
Using integer arrays to specify subscripts	N	
Using integer arrays to specify rank and bound of an array	N	
Using an integer constant to specify rank	N	
Reduction specifier for do concurrent	P	Syntax is accepted
Enumerations	N	

2.3.2 Fortran 2018

All features except those listed in the following table are supported. Almost all of the unsupported features are related to the multi-image execution.

Feature	Status	Comments
Asynchronous communication	P	Syntax is accepted
Teams	N	
Image failure	P	stat_failed_image is added
Form team statement	N	
Change team construct	N	
Coarrays allocated in teams	N	
Critical construct	N	
Lock and unlock statements	N	
Events	N	
Sync team construct	N	
Image selectors	N	
Intrinsic functions get_team and team_number	N	
Intrinsic function image_index	N	

Feature	Status	Comments
Intrinsic function num_images	N	
Intrinsic function this_image	N	
Intrinsic move_alloc extensions	P	
Detecting failed and stopped images	N	
Collective subroutines	N	
New and enhanced atomic subroutines	N	
Failed images and stat= specifiers	N	
Intrinsic function coshape	N	

2.3.3 Fortran 2008

All features except those listed in the following table are supported.

Feature	Status	Comments
Coarrays	N	Lowering and runtime support is not implemented
do concurrent	P	Sequential execution works. Parallel support in progress
Internal procedure as an actual argument or pointer target	Y	Current implementation requires stack to be executable. See Proposal

2.3.4 Fortran 2003

All features except those listed in the following table are supported.

Feature	Status	Comments
Parameterized Derived Types	P	PDT with length type parameters is not supported. See Proposal
Assignment to allocatable	P	Assignment to whole allocatable in FORALL is not implemented
Pointer Assignment	P	Polymorphic assignment in FORALL is not implemented
The VOLATILE attribute	P	VOLATILE in procedure interfaces is not implemented
Asynchronous input/output	P	IO will happen synchronously
MIN/MAX extensions for CHARACTER	P	Some variants are not supported

2.3.5 Fortran 95

All features are supported.

2.3.6 Fortran 90

All features are supported.

2.3.7 FORTRAN 77

All features are supported.

3. OpenMP Support

This section describes the OpenMP Support for ATfL.

3.1 Clang OpenMP support

Refer to the status page for details on Clang OpenMP support.

3.2 Flang OpenMP Support

This section outlines the OpenMP API features supported by Flang. It is intended as a general reference. For the most accurate information on unimplemented features, rely on the compiler's TODO or "Not Yet Implemented" messages, which are considered authoritative. With the exception of a few corner cases, Flang offers full support for OpenMP 2.5, and partial support for OpenMP 3.1 and OpenMP 4.0. The tables below outline the current status of OpenMP 4.0, 3.1, 3.0 feature support. Work is ongoing to add support for OpenMP 4.5 and newer versions; a support statement for these will be shared in the future.

The feature support information is provided as a table with three columns that are self explanatory. The Status column uses the letters P, Y, N for the implementation status:

- P : Partial. When the implementation is incomplete for a few cases
- Y : Yes. When the implementation is complete
- N : No. When the implementation is absent

Note: No distinction is made between the support in Parser/Semantics, MLIR, Lowering or the OpenMPIRBuilder.

3.2.1 OpenMP 4.0

The table below details the implementation status of OpenMP 4.0 features.

Feature	Status	Comments
proc_bind clause	Y	
simd construct	P	Some clauses are not supported
declare simd construct	N	
do simd construct	Y	
target data construct	N	
target construct	N	
target update construct	N	
declare target directive	N	

Feature	Status	Comments
teams construct	N	
distribute construct	N	
distribute simd construct	N	
distribute parallel loop construct	N	
distribute parallel loop simd construct	N	
depend clause	P	Depend clause with array sections are not supported
declare reduction construct	N	
atomic construct extensions	Y	
cancel construct	N	
cancellation point construct	N	
parallel do simd construct	Y	
target teams construct	N	
teams distribute construct	N	
teams distribute simd construct	N	
target teams distribute construct	N	
teams distribute parallel loop construct	N	
target teams distribute parallel loop construct	N	
teams distribute parallel loop simd construct	N	
target teams distribute parallel loop simd construct	N	

3.2.2 OpenMP 3.1, OpenMP 3.0

The table below details the implementation status of OpenMP 3.* features.

Feature	Status	Comments
intent(in) in firstprivate	Y	
pointers in firstprivate and lastprivate	Y	
final and mergeable clauses in task	Y	
taskyield construct	Y	
atomic construct extensions	Y	
assumed-size arrays are shared	Y	
allocatable arrays in private, firstprivate, lastprivate, reduction, copyin, copyprivate	Y	
firstprivate in default	Y	
collapse clause	Y	
schedule kind auto	Y	
task construct	P	delayed execution of tasks is not supported
taskwait construct	Y	

3.2.3 OpenMP 2.5, OpenMP 1.1

All features except a few corner cases in atomic (complex type, different but compatible types in lhs and rhs), threadprivate (character type) constructs/clauses are supported.

Proprietary Notice

This document is protected by copyright and other related rights and the use 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 Limited ("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 the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. 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, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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.

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 this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in Arm documents.

Product status

All products and services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

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

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
201-00	30 April 2025	Non-Confidential	Initial release

Change history

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in Document release information on page 28.

Table 2: Differences between issues

Change	Location
Updated with 201-00 release details	Release note document

Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<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	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Caution

We recommend the following. If you do not follow these recommendations your system might not work.

Warning

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or harming yourself.

This information is important and needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Arm documents are available on developer.arm.com/documentation.

Confidential documents are only available to licensees, when logged in. Each document link in the tables below provides direct access to the online version of the document.