



Get started with Arm Performance Libraries (Windows version)

Version 23.10

Non-Confidential

Copyright © 2023 Arm Limited (or its affiliates).
All rights reserved.

Issue 00

109361_2310_00_en



Get started with Arm Performance Libraries (Windows version)

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
2310-00	30 September 2023	Non-Confidential	Updates coinciding with Arm PL 23.10

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, 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.

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 affiliates) 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 <https://www.arm.com/company/policies/trademarks>.

Copyright © 2023 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|version 21.0)

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.

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>.

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.

Contents

- 1. Overview.....6
- 2. Installation.....7
- 3. Environment configuration.....8
- 4. Compile and test the examples.....9
- 5. Library selection.....14
- 6. Further information.....15

1. Overview

Arm Performance Libraries provides optimized standard core math libraries for high-performance computing applications on Arm processors. The library routines, which are available through both Fortran and C interfaces, cover the following functionality:

- BLAS - Basic Linear Algebra Subprograms (including XBLAS, the Extra Precise BLAS).
- LAPACK 3.11.0 - 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.

Arm Performance Libraries is built with OpenMP across many BLAS, LAPACK, FFT, and sparse routines in order to maximize your performance in multi-processor environments.

Arm Performance Libraries is available for Linux, macOS and Windows.

This tutorial describes how to get started with the version of Arm Performance Libraries for Windows. To learn about how to get started with the version of Arm Performance Libraries for Linux, see the [Get started with Arm Performance Libraries for Linux tutorial](#). To learn about how to get started with the version of Arm Performance Libraries for macOS, see the [Get started with Arm Performance Libraries for macOS tutorial](#).

2. Installation

The [learn.arm.com install guide for Arm Performance Libraries](https://learn.arm.com/install-guide-for-Arm-Performance-Libraries) covers the installation basics for all platforms.

Arm Performance Libraries can be downloaded from developer.arm.com.

Following installation you should have the environment variable `ARMPL_DIR` set to point to the directory in the Arm Performance Libraries installation which contains (amongst other things) the `include` and `lib` directories containing the header and library files.

3. Environment configuration

This section describes how to set up your environment before using Arm Performance Libraries with Windows.

Prerequisites

- You or your administrator has installed Arm Performance Libraries (see [Installation](#)).
- You have installed on your system either:
 - Microsoft Visual Studio, or
 - LLVM for Windows with `clang` C/C++ compiler and, for those with Fortran code to compile, `flang-new`.

See the Release Notes included in your installation for compiler version requirements associated with your release.

Setup

To configure your environment for Arm Performance Libraries:

Set the following environment variables:

```
set ARMPL_DIR=<location of armpl_<version> directory>  
set PATH=%PATH%;%ARMPL_DIR%\bin
```

See the [learn.arm.com install guide for Arm Performance Libraries](https://learn.arm.com/install-guide-for-Arm-Performance-Libraries) for instructions on how to set these environment variables via the Windows Control Panel.

4. Compile and test the examples

Arm Performance Libraries includes a number of example programs to compile and run.

The examples are located in `%ARMPL_DIR%\examples*`.

Multiple examples directories are provided in the installation. The suffix of the directory name indicates whether the examples inside link to the 32-bit (`_lp64`) or 64-bit (`_ilp64`) integer variants, and sequential (no suffix indicator) or OpenMP (`_mp`) multi-threaded variants, of Arm Performance Libraries.

For more information about the examples provided, see the [Arm Performance Libraries Reference Guide](#).

Each `examples*` directory contains the following:

- A `Makefile` to build and execute all of the example programs.
- A number of different C examples, `*.c`.
- A number of different Fortran examples, `*.f90`.
- Expected output for each example, `*.expected`.

The Makefile compiles and runs each example, and compares the generated output to the expected output. Any differences are flagged as errors.

Assuming you have first setup your environment to use Arm Performance Libraries (see [Environment configuration](#)), then to compile the C examples and run the tests:

1. Using the Windows File Explorer locate your installation of Arm Performance Libraries (`%ARMPL_DIR%`).
2. Copy one of the `examples*` folders somewhere writable.
3. Open a command prompt (search: `cmd`), then change into the `examples*` directory in the writeable location and run `nmake`:

```
cd path\to\examples*
nmake
```

The Makefile produces output similar to the following sample:

```
Compiling program armpl_dgemm_interleave_batch_c_example.c:

cl.exe -c /MD /nologo /I%ARMPL_DIR%\include_lp64
armpl_dgemm_interleave_batch_c_example.c /
Foarmpl_dgemm_interleave_batch_c_example.obj
armpl_dgemm_interleave_batch_c_example.c
Linking program armpl_dgemm_interleave_batch_c_example.exe:

cl.exe /MD armpl_dgemm_interleave_batch_c_example.obj C:\arm-
performance-libraries_23.10\armpl_23.10\lib\armpl_lp64_mp.dll.lib C:
\arm-performance-libraries_23.10\armpl_23.10\lib\FortranRuntime.lib
C:\arm-performance-libraries_23.10\armpl_23.10\lib\FortranDecimal.lib
```

```
C:\arm-performance-libraries_23.10\armpl_23.10\lib\omp.dll.lib /
Fearmpl_dgemm_interleave_batch_c_example.exe
Microsoft (R) Incremental Linker Version 14.36.32532.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:armpl_dgemm_interleave_batch_c_example.exe
armpl_dgemm_interleave_batch_c_example.obj
C:\arm-performance-libraries_23.10\armpl_23.10\lib\armpl_lp64_mp.dll.lib
C:\arm-performance-libraries_23.10\armpl_23.10\lib\FortranRuntime.lib
C:\arm-performance-libraries_23.10\armpl_23.10\lib\FortranDecimal.lib
C:\arm-performance-libraries_23.10\armpl_23.10\lib\omp.dll.lib

Running program armpl_dgemm_interleave_batch_c_example.exe with 4 threads:

    armpl_dgemm_interleave_batch_c_example.exe >
    armpl_dgemm_interleave_batch_c_example.res ;

ARMPL example: interleave-batch matrix multiplication
-----

blas_sum = 3.955600e+06
ib_sum = 3.955600e+06

----- Example
armpl_dgemm_interleave_batch_c_example.exe completed -----

Compiling program dgesdd_c_example.c:
...

All tests completed

-----
Success: All examples completed successfully
```

Example: fftw_dft_r2c_1d_c_example.c

The `fftw_dft_r2c_1d_c_example.c` example does the following:

- Creates an FFT plan for a one-dimensional, real-to-Hermitian Fourier transform, and a plan for its inverse, Hermitian-to-real transform.
- Executes the first plan to output the transformed values in `y`.
- Destroys the first plan.
- Prints the components of the transform.
- Executes the second plan to get the original data, unscaled.
- Destroys the second plan.
- Outputs the original and restored values, scaled (they should be identical).

```
/*
 * fftw_dft_r2c_1d: FFT of a real sequence
 *
 * ARMPL version 23.10 Copyright ARM 2023
 */

#include <armpl.h>
#include <fftw3.h>
#include <math.h>
#include <stdio.h>
```

```
#include "round_eps_to_zero.h"

int main(void) {
#define NMAX 20
    double xx[NMAX];
    double x[NMAX];
    // The output vector is of size (n/2)+1 as it is Hermitian
    fftw_complex y[NMAX / 2 + 1];

    printf("ARMPL example: FFT of a real sequence using fftw_plan_dft_r2c_1d\n");
    printf("-----\n");
    printf("\n");

    /* The sequence of double data */
    int n = 7;
    x[0] = 0.34907;
    x[1] = 0.54890;
    x[2] = 0.74776;
    x[3] = 0.94459;
    x[4] = 1.13850;
    x[5] = 1.32850;
    x[6] = 1.51370;

    // Use dcopy to copy the values into another array (preserve input)
    cblas_dcopy(n, x, 1, xx, 1);

    // Initialise a plan for a real-to-complex 1d transform from x->y
    fftw_plan forward_plan = fftw_plan_dft_r2c_1d(n, x, y, FFTW_ESTIMATE);
    // Initialise a plan for a complex-to-real 1d transform from y->x (inverse)
    fftw_plan inverse_plan = fftw_plan_dft_c2r_1d(n, y, x, FFTW_ESTIMATE);

    // Execute the forward plan and then deallocate the plan
    /* NOTE: FFTW does NOT compute a normalised transform -
     * returned array will contain unscaled values */
    fftw_execute(forward_plan);
    fftw_destroy_plan(forward_plan);

    printf("Components of discrete Fourier transform:\n");
    printf("\n");
    int j;
    for (j = 0; j <= n / 2; j++) {
        // Scale factor of 1/sqrt(n) to output normalised data
        double y_real = round_eps_to_zero_d(creal(y[j]) / sqrt(n));
        double y_imag = round_eps_to_zero_d(cimag(y[j]) / sqrt(n));
        printf("%4d    (%7.4f%7.4f)\n", j + 1, y_real, y_imag);
    }

    // Execute the reverse plan and then deallocate the plan
    /* NOTE: FFTW does NOT compute a normalised transform -
     * returned array will contain unscaled values */
    fftw_execute(inverse_plan);
    fftw_destroy_plan(inverse_plan);

    printf("\n");
    printf("Original sequence as restored by inverse transform:\n");
    printf("\n");
    printf("      Original    Restored\n");
    for (j = 0; j < n; j++) {
        double xx_j = round_eps_to_zero_d(xx[j]);
        // Scale factor of 1/n to output normalised data
        double x_j = round_eps_to_zero_d(x[j] / n);
        printf("%4d    %7.4f    %7.4f\n", j + 1, xx_j, x_j);
    }
    return 0;
}
```

To compile and run the example take a copy of the code from one of the examples directories and follow the steps below:

1. To generate an object file, compile the source `fftw_dft_r2c_1d_c_example.c`:

Compiler	Command
cl (MD)	<code>cl /MD /c /I%ARMPL_DIR%\include_lp64 /Ffftw_dft_r2c_1d_c_example.obj fftw_dft_r2c_1d_c_example.c</code>
cl (MT)	<code>cl /MT /c /I%ARMPL_DIR%\include_lp64 /Ffftw_dft_r2c_1d_c_example.obj fftw_dft_r2c_1d_c_example.c</code>
clang (MD)	<code>clang -fms-runtime-lib=dll -c -I%ARMPL_DIR%\include_lp64 -o fftw_dft_r2c_1d_c_example. obj fftw_dft_r2c_1d_c_example.c</code>
clang (MT)	<code>clang -fms-runtime-lib=static -c -I%ARMPL_DIR%\include_lp64 -o fftw_dft_r2c_1d_c_example.obj fftw_dft_r2c_1d_c_example.c</code>



clang compiler flags `-fms-runtime-lib=dll` and `-fms-runtime-lib=static` correspond to the `cl /MD` and `/MT` flags, respectively.



The FFTW interface, used in this example, defines its own complex types `fftw_complex` (double precision) and `fftwf_complex` (single precision). If you are using the Microsoft compiler `cl` these types are defined using the Microsoft complex types `_Dcomplex` and `_Fcomplex`, respectively. See [C complex math support](#) for more details about the Microsoft complex types. When using clang on Windows with Arm Performance Libraries, the FFTW complex types are instead defined using simple 2-element arrays to represent the complex types: `double [2]` and `float [2]`. The FFTW examples provided are set up to work with `cl` and use manipulation functions such as `creal` and `cimag` to extract the real and imaginary parts of a number. In order to get the Arm Performance Libraries FFTW examples to work with clang instead, you should replace calls with array manipulation instead. E.g. `creal(y[j])` in `fftw_dft_r2c_1d_c_example.c` becomes `y[j][0]` and `cimag(y[j])` becomes `y[j][1]`. For more information, see [the Arm Performance Libraries Reference Guide](#) where we discuss complex types on Windows.

2. Link the object code into an executable:

Compiler	Command
cl (MD)	<code>cl /MD fftw_dft_r2c_1d_c_example.obj /Ffftw_dft_r2c_1d_c_example.exe %ARMPL_DIR%\lib \armpl_lp64.lib %ARMPL_DIR%\lib\FortranRuntime.lib %ARMPL_DIR%\lib\FortranDecimal.lib</code>
cl (MT)	<code>cl /MT fftw_dft_r2c_1d_c_example.obj /Ffftw_dft_r2c_1d_c_example.exe %ARMPL_DIR%\lib \armpl_lp64.lib %ARMPL_DIR%\lib\FortranRuntime.lib %ARMPL_DIR%\lib\FortranDecimal.lib</code>
clang (MD)	<code>clang -fms-runtime-lib=dll fftw_dft_r2c_1d_c_example.obj -o fftw_dft_r2c_1d_c_example. exe %ARMPL_DIR%\lib\armpl_lp64.lib %ARMPL_DIR%\lib\FortranRuntime.lib %ARMPL_DIR%\lib \FortranDecimal.lib</code>

Compiler	Command
clang (MT)	clang -fms-runtime-lib=static fftw_dft_r2c_1d_c_example.obj -o fftw_dft_r2c_1d_c_example.exe %ARMPL_DIR%\lib\libarmpl_lp64.lib %ARMPL_DIR%\lib \libFortranRuntime.lib %ARMPL_DIR%\lib\libFortranDecimal.lib

The linker and compiler options are:

- /MD for c1 and -fms-runtime-lib=dll for clang: compile and link code that will use DLL versions of the Microsoft UCRT.
- /MT for c1 and -fms-runtime-lib=static for clang: compile and link code that will use static versions of the Microsoft UCRT.
- -I%ARMPL_DIR%\include_lp64 adds the Arm Performance Libraries location to the include directory search path.
- %ARMPL_DIR%\lib\armpl_lp64.lib links against Arm Performance Libraries (serial, 32-bit integer interfaces, /MD linkage).
- %ARMPL_DIR%\lib\libarmpl_lp64.lib links against Arm Performance Libraries (serial, 32-bit integer interfaces, /MT linkage).
- %ARMPL_DIR%\lib\FortranRuntime.lib %ARMPL_DIR%\lib\FortranDecimal.lib links against the Fortran runtime libraries that Arm Performance Libraries depends on (/MD linkage).
- %ARMPL_DIR%\lib\libFortranRuntime.lib %ARMPL_DIR%\lib\libFortranDecimal.lib links against the Fortran runtime libraries that Arm Performance Libraries depends on (/MT linkage).

3. Run the executable on your Arm system:

```
fftw_dft_r2c_1d_c_example.exe
```

The executable produces output as follows:

```
ARMPL example: FFT of a real sequence using fftw_plan_dft_r2c_1d
-----
Components of discrete Fourier transform:

 1 ( 2.4836 0.0000)
 2 (-0.2660 0.5309)
 3 (-0.2577 0.2030)
 4 (-0.2564 0.0581)

Original sequence as restored by inverse transform:

      Original   Restored
 1    0.3491    0.3491
 2    0.5489    0.5489
 3    0.7478    0.7478
 4    0.9446    0.9446
 5    1.1385    1.1385
 6    1.3285    1.3285
 7    1.5137    1.5137
```

5. Library selection

Arm Performance Libraries contains multiple different types of library. Your installation contains both dynamic and static libraries, and, in each case, there are serial and multi-threaded libraries. Furthermore, for each of those combinations there are also libraries which take 32-bit integer arguments in function interfaces, and also libraries which take 64-bit integer arguments.

Here we show the options needed to use the different types of library.

Compile	Link	Description
/I%ARMPL_DIR\include_lp64	%ARMPL_DIR\lib\armpl_lp64.dll.lib	Use 32-bit integers, single-threaded library.
/DINTEGER64 /I%ARMPL_DIR\include_ilp64	%ARMPL_DIR%\lib\armpl_ilp64.dll.lib	Use 64-bit integers, single-threaded library.
/I%ARMPL_DIR\include_lp64	%ARMPL_DIR\lib\armpl_lp64_mp.dll.lib	Use 32-bit integers, multi-threaded (OpenMP) library.
/DINTEGER64 /I%ARMPL_DIR\include_ilp64	%ARMPL_DIR%\lib\armpl_ilp64_mp.dll.lib	Use 64-bit integers, multi-threaded (OpenMP) library.

Linking against static libraries

The libraries are supplied in both static and dynamic (DLL) versions. The commands given above link to the dynamic versions of the libraries. To force linking with static versions of the libraries, prefix the library names with `lib` and remove `.dll`. For example, `libarmpl_lp64_mp.lib` is the static 32-bit integer multi-threaded library. For more information about linkage on Windows, see the [Windows UCRT linkage section](#) of the Arm Performance Libraries Reference Guide.



Note

The compiler flag distinction given in the tables between using the `include_lp64` directory for 32-bit integer interfaces and using `include_ilp64` for 64-bit integers only actually matters for users of the interface blocks in the Fortran pre-compiled modules (e.g. `armpl_blas.mod`). For C users and Fortran users who are not using the interface blocks (or are re-compiling the modules from source), it does not matter which include directory you choose to use. For more information, see the [Arm Performance Libraries Reference Guide](#).

6. Further information

The following links contain detailed documentation about different aspects of using Arm Performance Libraries:

- The [learn.arm.com install guide](https://learn.arm.com/install-guide) shows how to install Arm Performance Libraries on all supported platforms.
- See the [developer.arm.com downloads page for stand-alone versions of Arm Performance Libraries](https://developer.arm.com/downloads-page-for-stand-alone-versions-of-Arm-Performance-Libraries) for the full list of supported platforms.
- Arm Compiler for Linux, which includes Arm Performance Libraries, can also be downloaded from developer.arm.com.
- [Arm Performance Libraries Reference Guide](#) provides comprehensive documentation for all functions.
- If you have any questions or queries about using Arm Performance libraries please post a message on the [Compilers and Libraries support forum](#). See below for guidance on how to do this effectively.

Reporting issues

To get help with any issue that you are experiencing, it helps to report information about the version of Arm Performance Libraries that you are using and the system that you are running on.

You can obtain the necessary system and library information by running the `armpl-info.exe` program. You can find the `armpl-info.exe` file in the `%ARMPL_DIR%\bin` directory of your installation. Execute the program to see the information printed to the screen:

```
%ARMPL_DIR%\bin\armpl-info.exe
```

Other releases of Arm Performance Libraries

Arm Performance Libraries is also available:

- [As part of Arm Compiler for Linux.](#)
- [As stand-alone Linux releases, compatible with GCC and NVHPC.](#)
- [For macOS, compatible with LLVM.](#)