Get started with Arm Performance Libraries (standalone version)

0100-00

Non-Confidential

Issue 00

109408_0100_00_en

Get started with Arm Performance Libraries (standalone version)

Release information

Document history

Issue	Date	Confidentiality	Change
0100-00	16 October 2023	Non-Confidential	Document Migration

Proprietary Notice

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

Arm Performance Libraries provide 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, include:

- BLAS - Basic Linear Algebra Subprograms (including XBLAS, the extended precision BLAS).
- LAPACK 3.10.0 - 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.
- libamath - a subset of libm, which is a set of optimized mathematical functions.
- libastring - a subset of libc, which is a set of optimized string functions.

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

2. Installation

This guide contains instructions to install on the following operating systems:

- Install on Linux
- Install on Apple MacOS
- Install on Windows

Note: To use Arm Performance Libraries functions in your code, you must include the header file <armpl.h>. This header file is located in /opt/arm/<armpl_dir>/include/, or <install_dir>/<armpl_dir>/include/ if you have installed to a different location than the default. If you use FFTs, you will also need to include the `fftw3.h` header file. If you include other legacy header files such as `blas.h` or `lapack.h`, they will also work.

2.1 Install on Linux

To install Arm Performance Libraries:

1. Extract the downloaded package and change directory into the resulting directory:

   ```
   tar -xvf <package_name>.tar
   cd <package_name>
   ```

   Replace <package_name> with the full name of the downloaded package.

2. Run the installation script as a privileged user and pass any options to configure the installation:

   ```
   ./arm-performance-libraries_<version>_*.sh <options>
   ```

   Where `<options>` can be one or more of:

Option	Description
`-a, -accept`	Automatically accept the EULA (the EULA still displays).
`-l, -list-packages`	List the installer packages
`-i, -install-to <location>`	Install to the given directory. Use this option if you do not have sudo rights to install to /opt/arm or another central location.
`-s, -save-packages-to <location>`	Save packages to given directory.
`-f, -force`	Force an install attempt to a non empty directory.
`-h, -help`	Display this table in the form of a help message in the terminal

3. The installer displays the EULA and prompts you to agree to the terms. Type ‘yes’ at the prompt to continue

The packages are installed to `<install_dir>/<package_name>`, where `<install_dir>` defaults to /opt/arm if not explicitly provided using the –install-to option.

2.2 Install on Apple MacOS

To install Arm Performance Libraries:

1. Mount the disk image by double clicking the icon of the downloaded package, or by running from a terminal:

```
hdiutil attach <package_name>.dmg
```

   Replace `<package_name>` with the name of the downloaded package.

2. Run the installation script as a privileged user and pass any options to configure the installation:

```
/Volumes/<package_name>_installer/<package_name>_install.sh <options>
```

   `<options>` can be one or more of:

Option	Description
`-y`	Automatically accept the EULA (the EULA does not display).
`—install_dir=<location>`	Install to the given directory. Use this option if you do not have required privileges.
`-h`	Display this table in the form of a help message in the terminal.

3. The installer displays the EULA and prompts you to agree to the terms. Type 'y' at the prompt to continue.

   The packages are installed to `<install_dir>/<package_name>`, where `<install_dir>` defaults to `/opt/arm` if not explicitly provided using the `--install_dir` option.

   in case you selected -y and want to read the EULA later, it is available at:

```
<install_dir>/arm-performance-libraries_<version>/license_terms/
license_agreement.txt
```

2.3 Install on Windows

To install Arm Performance Libraries:

1. Unpack and extract the zip file:

- Locate the downloaded zip file in the Windows File Explorer.
- Double click on the file, and then click on the “Extract all” button at the top of the File Explorer.
- In the pop-up window, select a location to unpack Arm Performance Libraries into on your system, and click on Extract.

2. Update your environment variables:
    - Using the Windows Search feature, search for “System Properties”. Click on “Best Match”.
    - In the System Properties window click on “Environment Variables”.
    - Add a new environment variable called ARMPL_DIR, which should have the value

```
<install location>\arm-performance-libraries_23.10\armpl_23.10
```

where `<install location>` is the location in which you unpacked Arm Performance Libraries in step 1 above.

- Next, add `%ARMPL_DIR%\bin` to the list of directories in the PATH environment variable.
- Read the enclosed Release Notes and examples to find out how to link your application to the library in a command terminal. Users wishing to use Arm PL from within MSVC are referred to Microsoft’s online documentation.

3. Environment configuration

This section describes how to load the correct environment module for Arm Performance Libraries.

Procedure

Use the following steps to load the Arm Performance Libraries module:

1. Use this command to see which environment modules are available:

   ```
   module avail
   ```

   Note: You might need to configure the `MODULEPATH` environment variable to include the installation directory:

   ```
   export MODULEPATH=$MODULEPATH:/opt/arm/modulefiles/
   ```

2. Load the appropriate module for the OS and version of GCC that you are using.

   For example:

   ```
   module load armpl/22.0_gcc-11.2
   ```

   Tip: Consider adding the module load command to your .profile to run it automatically every time you log in.

4. Compile and test the examples

Arm Performance Libraries include a number of example programs to compile and run. The examples are located in `/opt/arm/<armpl_dir>/examples/`, or `<install_dir>/<armpl_dir>/examples/`, if you have installed to a different location than the default.

The `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.

To compile the examples and run the tests, use the following command:

```
make
```

The Makefile produces output similar to the following sample:

```
Compiling program armplinfo.f90:
gfortran -c -mcpu=native -I../include armplinfo.f90 -o armplinfo.o
Linking program armplinfo.exe:
gfortran -mcpu=native armplinfo.o -L../lib -larmpl -lm -o armplinfo.exe
Running program armplinfo.exe:
LD_LIBRARY_PATH=/opt/arm/arm-linux-compiler-0.0_Generic-AArch64_RHEL-8_aarch64-
linux/lib ./armplinfo.exe > armplinfo.res
ARMPL (ARM Performance Libraries)

...

Testing: no example difference files were generated.
Test passed OK
```

4.1 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 22.0 Copyright Arm 2022
 */
#include <armpl.h>
#include <complex.h>
#include <fftw3.h>
#include <math.h>
#include <stdio.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
                printf("%4d   (%7.4f%7.4f)\n", j + 1, creal(y[j]) / sqrt(n),
                       cimag(y[j]) / sqrt(n));
        // 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++)
                // Scale factor of 1/n to output normalised data
                printf("%4d   %7.4f   %7.4f\n", j + 1, xx[j], x[j] / n);
        return 0;
}
```

To compile and run the example take a copy of the code from `<install-dir>/examples` and follow the steps below:

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

Compiler	Command
gcc	gcc fftw_dft_r2c_1d_c_example.o -L<install_dir>/lib -o fftw_dft_r2c_1d_c_example.exe -larmpl_lp64 -lm

2. Link the object into an executable:

Compiler	Command
gcc	gcc fftw_dft_r2c_1d_c_example.o -L<install_dir>/lib -o fftw_dft_r2c_1d_c_example.exe -larmpl_lp64 -lm

The linker and compiler options are:

- `-L<install_dir>/lib` adds the Arm Performance Libraries location to the library search path.
- `-larmpl_lp64` links against Arm Performance Libraries.
- `-lm` links against the standard math libraries.

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. Optimized math routines - libamath

libamath contains AArch64-optimized versions of the following scalar functions, in both single and double precision: exponential (`exp, exp2`), logarithm (`log, log2, log10`), and error functions (`erf, erfc`). In addition, optimized single precision sine and cosine functions are included (`sinf, cosf, sincosf`).

libamath also contains vectorized versions (Neon and SVE) of all of the common `math.h` functions in libm.

You must explicitly link to the libamath library before linking to libm. For example:

```
gcc code_with_math_routines.c -lamath -lm
```

```
gfortran code_with_math_routines.f -lamath -lm
```

6. Optimized string routines - libastring

libastring provides a set of replacement string.h functions which are optimized for AArch64: `bcmp`, `memchr`, `memcpy`, `memmove`, `memset`, `strchr`, `strchrnul`, `strcmpstrcpy`, `strlen`, `strncmp`, `strnlen`.

You must explicitly link to the libastring library to benefit from the performance increase. For example:

```
gcc code_with_string_routines.c -lastring
```

```
gfortran code_with_string_routines.f -lastring
```

7. Library selection

To instruct your compiler to load the optimum version of Arm Performance Libraries for your target architecture and implementation, you can use `-larmpl` option.

Supported options and arguments are:

GCC flag	Description
`-DINTEGER32` (Compile) `-larmpl_lp64` (Link)	Use 32-bit integers.
`-DINTEGER32` (Compile) `-larmpl_lp64` (Link)	Use 64-bit integers.
`-larmpl_lp64`	Use the single-threaded library.
`-larmpl_lp64_mp`	Use the OpenMP multi-threaded library.

7.1 Linking against static libraries

The Arm Performance Libraries are supplied in both static and shareable versions, `libarmpl_lp64.a` and `libarmpl_lp64.so`. By default, the commands given above link to the shareable version of the library, `libarmpl_lp64.so`, if that version exists in the specified directory.

To force linking to the static library, add the `-static` option.

8. Documentation

The Arm Performance Libraries Reference Guide is available on the Arm Developer website.