AMD CPU Libraries User Guide

Version 1.3

Contents

1. Introduction

AMD CPU Libraries are a set of numerical libraries optimized for AMD EPYC^{TM} processor family. This document provides instructions on installing and using all the AMD optimized libraries.

AMD CPU Libraries comprise of six packages, primarily,

1. **BLIS (BLAS Library)** – BLIS is a portable open-source software framework for instantiating high-performance Basic Linear Algebra Subprograms (BLAS) functionality.
2. **libFLAME (LAPACK)** - libFLAME is a portable library for dense matrix computations, providing much of the functionality present in Linear Algebra Package (LAPACK).
3. **AMD Random Number Generator Library** - AMD Random Number Generator Library is a pseudorandom number generator library
4. **AMD Secure RNG** - The AMD Secure Random Number Generator (RNG) is a library that provides APIs to access the cryptographically secure random numbers generated by AMD's hardware random number generator implementation.
5. **LibM (AMD Core Math Library)** - AMD LibM is a software library containing a collection of basic math functions optimized for x86-64 processor based machines.
6. **FFTW** - We recommend **FFTW (Fast Fourier Transform in the West)**, the popular open source FFT library for AMD EPYC^{TM} processor. FFTW is a comprehensive collection of fast C routines for computing the Discrete Fourier Transform (DFT) and various special cases thereof.

Latest information on the AMD Libraries release and installers are available in the following AMD developer site. https://developer.amd.com/amd-cpu-libraries/. For any issues or queries regarding the libraries, please contact cpulibsupport@amd.com

2. BLIS

BLIS is a portable open-source software framework for instantiating high-performance Basic Linear Algebra Subprograms (BLAS) - like dense linear algebra libraries. The framework was designed to isolate essential kernels of computation that, when optimized, immediately enable optimized implementations of most of its commonly used and computationally intensive operations. Select kernels have been optimized for the AMD EPYC^{TM} processor family by AMD and others

2.1. Installation

BLIS can be installed either from source or pre-built binaries

2.1.1. Build BLIS from source

Github link: https://github.com/amd/blis

2.1.1.1. Single-thread BLIS

Here are the build instructions for single threaded AMD BLIS.

1. git clone https://github.com/amd/blis.git

2. Depending on the target system, and build environment, one would have to enable/disable suitable configure options. The following steps provides instructions for compiling on AMD "Zen" core based platforms. For a complete list of options and their description, type ./configure –help.

   $ ./configure --enable-cblas --prefix=*<your-install-dir>* zen

3. $ make

4. $ make install

2.1.1.2. Multi-threaded BLIS

Here are the build instructions for multi-threaded AMD BLIS.

1. git clone https://github.com/amd/blis.git

2. Depending on the target system, and build environment, one would have to enable/disable suitable configure options. The following steps provides instructions for compiling on AMD "Zen" core based platforms. For a complete list of options and their description, type ./configure –help.

   $ ./configure --enable-cblas --enable-threading=[Mode] --prefix=*<your-install-dir>* zen

   [Mode] values can be openmp,pthread,no. "no" will disable multi-threading.

Example to use OpenMP threading model:

$ ./configure --enable-cblas --enable-threading=openmp --prefix=<*your-install-dir*> zen

2. $ make

3. $ make install

2.1.2. Using pre-built binaries

AMD optimized BLIS libraries can be found in the following links. The binary packages for Ubuntu 18.04 LTS and CentOS 7.5 are available at the location. For other Operating systems, you can build from the source as explained in the previous section.

https://github.com/amd/blis/releases
https://developer.amd.com/amd-cpu-libraries/blas-library/

Also, BLIS binary can be installed from the tar file available in the following link. The tar file includes pre-built binaries of other AMD Libraries libFLAME, LibM, Random Number Generator and AMD Secure RNG

https://developer.amd.com/amd-cpu-libraries/

2.2.   Usage

To use BLIS in your application, you just need to link the library while building the application

Ex:

With Static Library:

```
gcc test_blis.c -I<path-to-BLIS-header>  <path-toBLIS-library>/libblis.a -o test_blis.x
```

With Dynamic Library:

```
gcc test_blis.c -I<path-to-BLIS-header>  -L<path-toBLIS-library>/libblis.so -o test_blis.x
```

BLIS also includes a BLAS compatibility layer which gives application developers access to BLIS implementations via traditional FORTRAN BLAS API calls, that can be used in FORTRAN as well as C code. BLIS also provides a CBLAS API, which is a C-style interface for BLAS, that can be called from C code.

2.2.1.   BLIS Usage in FORTRAN

BLIS can be used with FORTRAN applications through the standard BLAS API.

For example, see below, FORTRAN code that does double precision general matrix-matrix multiplication. It calls the 'DGEMM' BLAS API function to accomplish this. An example command to compile it and link with the BLIS library is also shown below the code.

```fortran
! File: BLAS_DGEMM_usage.f
! Example code to demonstrate BLAS DGEMM usage

program dgemm_usage

implicit none

EXTERNAL DGEMM

DOUBLE PRECISION, ALLOCATABLE :: a(:,:)
DOUBLE PRECISION, ALLOCATABLE :: b(:,:)
DOUBLE PRECISION, ALLOCATABLE :: c(:,:)
INTEGER I, J, M, N, K, lda, ldb, ldc
DOUBLE PRECISION alpha, beta

M=2
N=M
K=M
lda=M
ldb=K
ldc=M
alpha=1.0
beta=0.0

ALLOCATE(a(lda,K), b(ldb,N), c(ldc,N))

a=RESHAPE((/ 1.0, 3.0, &
             2.0, 4.0  /), &
             (/lda,K/))
b=RESHAPE((/ 5.0, 7.0, &
             6.0, 8.0  /), &
             (/ldb,N/))

WRITE(*,*) ("a =")
DO I = LBOUND(a,1), UBOUND(a,1)
    WRITE(*,*) (a(I,J), J=LBOUND(a,2), UBOUND(a,2))
END DO
WRITE(*,*) ("b =")
DO I = LBOUND(b,1), UBOUND(b,1)
    WRITE(*,*) (b(I,J), J=LBOUND(b,2), UBOUND(b,2))
END DO

CALL DGEMM('N','N',M,N,K,alpha,a,lda,b,ldb,beta,c,ldc)

WRITE(*,*) ("c =")
DO I = LBOUND(c,1), UBOUND(c,1)
    WRITE(*,*) (c(I,J), J=LBOUND(c,2), UBOUND(c,2))
END DO

end program dgemm_usage
```

Example compilation command, for the above code, with gfortran compiler:

```
gfortran -ffree-form BLAS_DGEMM_usage.f path/to/libblis.a
```

2.2.2. BLIS Usage in C through BLAS and CBLAS APIs

There are multiple ways to use BLIS with an application written in C. While one can always use the native BLIS API for the same, BLIS also includes BLAS and CBLAS interfaces.

Using BLIS with BLAS API in C code

Shown below is the C version of the code listed above in FORTRAN. It uses the standard BLAS API. Note that (a) the matrices are transposed to account for the row-major storage of C and the column-major convention of BLAS (inherited from FORTRAN), (b) the function arguments are passed by address, again to be in line with FORTRAN conventions, (c) there is a trailing underscore in the function name ('dgemm_'), as BLIS' BLAS APIs expect that (FORTRAN compilers add a trailing underscore), and (d) "blis.h" is included as a header. An example command to compile it and link with the BLIS library is also shown below the code.

```c
// File: BLAS_DGEMM_usage.c
// Example code to demonstrate BLAS DGEMM usage

#include<stdio.h>
#include "blis.h"

#define DIM 2

int main() {

        double a[DIM * DIM] = { 1.0, 3.0, 2.0, 4.0 };
        double b[DIM * DIM] = { 5.0, 7.0, 6.0, 8.0 };
        double c[DIM * DIM];
        int I, J, M, N, K, lda, ldb, ldc;
        double alpha, beta;

        M = DIM;
        N = M;
        K = M;
        lda = M;
        ldb = K;
        ldc = M;
        alpha = 1.0;
        beta = 0.0;

        printf("a = \n");
        for ( I = 0; I < M; I ++ ) {
                for ( J = 0; J < K; J ++ ) {
                        printf("%f\t", a[J * K + I]);
                }
                printf("\n");
        }
        printf("b = \n");
        for ( I = 0; I < K; I ++ ) {
                for ( J = 0; J < N; J ++ ) {
                        printf("%f\t", b[J * N + I]);
                }
                printf("\n");
        }
```

```
        dgemm_("N","N",&M,&N,&K,&alpha,a,&lda,b,&ldb,&beta,c,&ldc);

        printf("c = \n");
        for ( I = 0; I < M; I ++ ) {
                for ( J = 0; J < N; J ++ ) {
                        printf("%f\t", c[J * N + I]);
                }
                printf("\n");
        }

        return 0;
}
```

Example compilation command, for the above code, with gcc compiler:

```
gcc BLAS_DGEMM_usage.c -Ipath/to/include/blis/ path/to/libblis.a
```

Using BLIS with CBLAS API

The C code below shows using CBLAS APIs for the same functionality listed above. Note that (a) CBLAS Layout option allows us to choose between row-major and column-major layouts (row-major layout is used in the example, which is in line with C-style), (b) the function arguments can be passed by value also, and (c) "cblas.h" is included as a header. An example command to compile it and link with the BLIS library is also shown below the code. Also, note that, in order to get CBLAS API with BLIS, one has to supply the flag '--enable-cblas' to the 'configure' command while building the BLIS library.

```c
// File: CBLAS_DGEMM_usage.c
// Example code to demonstrate CBLAS DGEMM usage
#include<stdio.h>
#include "cblas.h"

#define DIM 2

int main() {
        double a[DIM * DIM] = { 1.0, 2.0, 3.0, 4.0 };
        double b[DIM * DIM] = { 5.0, 6.0, 7.0, 8.0 };
        double c[DIM * DIM];
        int I, J, M, N, K, lda, ldb, ldc;
        double alpha, beta;

        M = DIM;
        N = M;
        K = M;
        lda = M;
        ldb = K;
        ldc = M;
        alpha = 1.0;
        beta = 0.0;

        printf("a = \n");
        for ( I = 0; I < M; I ++ ) {
                for ( J = 0; J < K; J ++ ) {
                        printf("%f\t", a[I * K + J]);
                }
                printf("\n");
        }
        printf("b = \n");
        for ( I = 0; I < K; I ++ ) {
                for ( J = 0; J < N; J ++ ) {
                        printf("%f\t", b[I * N + J]);
                }
                printf("\n");
        }

        cblas_dgemm(CblasRowMajor,  CblasNoTrans, CblasNoTrans, M, N, K, alpha, a,
lda, b, ldb, beta, c, ldc);

        printf("c = \n");
        for ( I = 0; I < M; I ++ ) {
                for ( J = 0; J < N; J ++ ) {
                        printf("%f\t", c[I * N + J]);
                }
                printf("\n");
        }

        return 0;
}
```

Example compilation command, for the above code, with gcc compiler:

```
gcc CBLAS_DGEMM_usage.c -Ipath/to/include/blis/ path/to/libblis.a
```

3. libFLAME

libFLAME is a portable library for dense matrix computations, providing much of the functionality present in Linear Algebra Package (LAPACK). It includes a compatibility layer, FLAPACK, which includes complete LAPACK implementation. The library provides scientific and numerical computing communities with a modern, high-performance dense linear algebra library that is extensible, easy to use, and available under an open source license. libFLAME is a C-only implementation and does not depend on any external FORTRAN libraries including LAPACK. There is an optional backward compatibility layer, lapack2flame that maps LAPACK routine invocations to their corresponding native C implementations in libFLAME. This allows legacy applications to start taking advantage of libFLAME with virtually no changes to their source code.

In combination with BLIS library which includes optimizations for the AMD EPYC^{TM} processor family, libFLAME enables running high performing LAPACK functionalities on AMD platform.

3.1. Installation

libFLAME can be installed either from source or pre-built binaries

3.1.1. Build libFLAME from source

Github link: https://github.com/amd/libflame

Note: Building libFLAME library does not require linking to BLIS or any other BLAS library. Applications which use libFLAME will have to link with BLIS (or other BLAS libraries) for BLAS functionalities.

1. git clone https://github.com/amd/libflame.git

2. Run configure script. Example below shows few sample options. Enable/disable other flags as needed

    $ ./configure --enable-lapack2flame --enable-cblas-interfaces --enable-dynamic-build --prefix=*<your-install-dir>*

    'Argument list too long' build error
    On some build environments, compiling libFLAME might throw '*Argument list too long*' error message when *make* tries to archive the object files to library. In such case, in order to reduce the length of argument list, enable the option "--enable-max-arg-list-hack"

    $ ./configure --enable-lapack2flame --enable-cblas-interfaces --enable-dynamic-build ***--enable-max-arg-list-hack*** --prefix=*<your-install-dir>*

3. Make and install. By default the library will be installed to $HOME/flame
    - $ make
    - $ make install

3.1.2.   Using pre-built binaries

AMD optimized BLIS libraries can be found in the following links. The binary packages for Ubuntu 16.04 LTS are available at the location. For other Operating systems, you can build from the source as explained in the previous section.

https://github.com/amd/libflame/releases
https://developer.amd.com/amd-cpu-libraries/blas-library/#libflame

Also, libFLAME binary can be installed from the tar file available in the following link. The tar file includes pre-built binaries of other AMD Libraries BLIS, LibM, Random Number Generator and AMD Secure RNG

https://developer.amd.com/amd-cpu-libraries/

3.2.   Usage

Run libFLAME tests. Link with AMD optimized BLIS library

From the libFLAME source directory,

  o $ cd test

Create a directory for object files

  o $ mkdir obj

Edit the Makefile.Change LDFLAGS as

```
 LDFLAGS := -lm -lpthread
```
Modify the definition of LIBBLAS as : LIBBLAS :=*<path-to-BLIS-library>*

Comment out LIBLAPACK_PATH

Change LIBLAPACK to

```
  LIBLAPACK :=
```
Save the Makefile

```
 * $ make
 * $ ./test_libflame.x
```

4. AMD LibM

AMD LibM is a software library containing a collection of basic math functions optimized for x86-64 processor based machines. It provides many routines from the list of standard C99 math functions. It includes scalar as well as vector variants of the core math functions. AMD LibM is a C library, which users can link in to their applications to replace compiler-provided math functions. Applications can link into AMD LibM library and invoke math functions instead of compiler's math functions for better accuracy and performance.

Latest AMD LibM includes the 'alpha version' of vector variants for the core math functions; power, exponential, logarithmic and trigonometric. Few caveats of the vector variants are listed below.

- Vector variants are relaxed versions of the respective math functions.
- The routines take advantage of the AMD64 architecture for performance. Some of the performance is gained by sacrificing error handling or the acceptance of certain arguments.
- Denormal inputs may produce unpredictable results. It is therefore the responsibility of the caller of these routines to ensure that their arguments are suitable.
- The vector routines will have to be invoked using C intrinsics or from x86 assembly.

The list of available vector functions is given below. All functions have an 'amd_' prefix.

Exponential
  * vrs4_expf, vrs4_exp2f, vrs4_exp10f, vrs4_expm1f
  * vrsa_expf, vrsa_exp2f, vrsa_exp10f, vrsa_expm1f
  * vrd2_exp, vrd2_exp2, vrd2_exp10, vrd2_expm1, vrd4_exp, vrd4_exp2
  * vrda_exp, vrda_exp2, vrda_exp10, vrda_expm1

Logarithmic
  * vrs4_logf, vrs4_log2f, vrs4_log10f, vrs4_log1pf
  * vrsa_logf, vrsa_log2f, vrsa_log10f, vrsa_log1pf
  * vrd2_log, vrd2_log2, vrd2_log10, vrd2_log1p, vrd4_log
  * vrda_log, vrda_log2, vrda_log10, vrda_log1p

Trigonometric
  * vrs4_cosf, vrs4_sinf
  * vrsa_cosf, vrsa_sinf
  * vrd2_cos, vrd2_sin, vrd2_cosh, vrd2_sincos
  * vrda_cos, vrda_sin

 Power
   * vrs4_cbrtf, vrd2_cbrt, vrs4_powf, vrd2_pow, vrd4_pow
  * vrsa_cbrtf, vrda_cbrt, vrsa_powf

The scalar functions listed below are present in the library. They all have an 'amd_' prefix.

Trigonometric
* cosf, cos, sinf, sin, tanf, tan, sincosf, sincos
* acosf, acos, asinf, asin, atanf, atan, atan2f, atan2

Hyperbolic
* coshf, cosh, sinhf, sinh, tanhf, tanh
* acoshf, acosh, asinhf, asinh, atanhf, atanh

Exponential & Logarithmic
* expf, exp, exp2f, exp2, exp10f, exp10, expm1f, expm1
* logf, log, log10f, log10, log2f, log2, log1pf, log1p
* logbf, logb, ilogbf, ilogb
* modff, modf, frexpf, frexp, ldexpf, ldexp
* scalbnf, scalbn, scalblnf, scalbln

Power & Absolute value
* powf, pow, fastpow, cbrtf, cbrt, sqrtf, sqrt, hypotf, hypot
* fabsf, fabs

Nearest integer
* ceilf, ceil, floorf, floor, truncf, trunc
* rintf, rint, roundf, round, nearbyintf, nearbyint
* lrintf, lrint, llrintf, llrint
* lroundf, lround, llroundf, llround

Remainder
* fmodf, fmod, remainderf, remainder

Manipulation
* copysignf, copysign, nanf, nan, finitef, finite
* nextafterf, nextafter, nexttowardf, nexttoward

Maximum, Minimum & Difference
* fdimf, fdim, fmaxf, fmax, fminf, fmin

The declarations for all of these functions are as specified in the C99 standard. They are equivalent to the declarations present in the standard math header 'math.h'. The only difference is that the functions in AMD LibM have 'amd_' prefix in their names.

4.1.   Installation

AMD LibM binary can be downloaded from following link. The binary packages for Ubuntu 16.04 LTS are available at the location.

https://developer.amd.com/amd-cpu-libraries/amd-math-library-libm/

Also, LibM binary can be installed from the tar file available in the following link. The tar file includes pre-built binaries of other AMD Libraries BLIS, libFLAME, Random Number Generator and AMD Secure RNG

https://developer.amd.com/amd-cpu-libraries/

4.2.   Usage

In order to use AMD LibM in your application, follow the below steps.

- Include 'amdlibm.h' in your C/C++ code and call the desired math function
- If math.h has to be included, include math.h before amdlibm.h
- Link in the appropriate version of the library in your program

The Linux libraries have a dependency on system math library. When linking AMD LibM, ensure it precedes system math library in the link order i.e., "-lamdlibm" should come before "-lm". Explicit linking of system math library is required when using gcc C compiler. With g++ compiler (for C++), this is not needed.

A simple trick can be used to replace existing math function calls in order to use AMD LibM. To replace a single function, for example, to replace all 'sin' function calls in existing code, use the following construct in your source code.

```
#undef sin
#define sin amd_sin
```

In order to replace all math functions to that of AMD LibM, define 'REPLACE_WITH_AMDLIBM' preprocessor macro.

Refer to the examples directory under the LibM install location for illustration.

5. AMD Random Number Generator

AMD Random Number Generator Library is a pseudorandom number generator library. It provides a comprehensive set of statistical distribution functions and various uniform distribution generators (base generators) including Wichmann-Hill and Mersenne Twister. The library contains five base generators and twenty-three distribution generators. In addition, users can supply a custom-built generator as the base generator for all the distribution generators.

5.1. Installation

AMD Random Number Generator binary can be downloaded from following link. The binary packages for Ubuntu 16.04 LTS are available at the location.

https://developer.amd.com/amd-cpu-libraries/rng-library/

Also, the Random Number Generator binary can be installed from the tar file available in the following link. The tar file includes pre-built binaries of other AMD Libraries BLIS, libFLAME, LibM and AMD Secure RNG

https://developer.amd.com/amd-cpu-libraries/

5.2. Usage

To use AMD Random Number Generator library in your application, you just need to link the library while building the application

Following is a sample Makefile for an application that uses AMD Random Number Generator library.

```
RNGDIR := <path-to-Random-Number-Generator-library>

CC := gcc

CFLAGS := -I$(RNGDIR)/include

CLINK := $(CC)

CLINKLIBS := -lgfortran -lm -lrt -ldl

LIBRNG := $(RNGDIR)/lib/librng_amd.so

//Compile the program

$(CC) -c $(CFLAGS) test_rung.c -o test_rng.o

//Link the library

$(CLINK) test_rng.o $(LIBRNG) $(CLINKLIBS) -o test_rng.exe
```

Refer to the examples directory under the AMD Random Number Generator library install location for illustration.

6. AMD Secure RNG

The AMD Secure Random Number Generator (RNG) is a library that provides APIs to access the cryptographically secure random numbers generated by AMD's hardware-based random number generator implementation. These are highly quality robust random numbers designed to be suitable for cryptographic applications. The library makes use of RDRAND and RDSEED x86 instructions exposed by the AMD hardware. Applications can just link to the library and invoke either a single or a stream of random numbers. The random numbers can be of 16-bit, 32-bit, 64-bit or arbitrary size bytes.

6.1.   Installation

AMD Secure RNG library can be downloaded from following link.

https://developer.amd.com/amd-cpu-libraries/rng-library/

Also, AMD Secure RNG can be installed from the tar file available in the following link. The tar file includes pre-built binaries of other AMD Libraries BLIS, libFLAME, LibM and AMD Random Number Generator library.

https://developer.amd.com/amd-cpu-libraries/

6.2.   Usage

Following are the source files included in the AMD Secure RNG package

1.   include/secrng.h : Header file that has declaration of all the library APIs.
2.   src_lib/secrng.c : Has the implementation of the APIs
3.   src_test/secrng_test.c : Test application to test all the library APIs
4.   Makefile : To compile the library and test application

Application developers can use the included makefile to compile the source files and generate dynamic and static libraries. They can then link it to their application and invoke the required APIs.

Below code snippet shows sample usage of the library API. In this example, get_rdrand64u is invoked to return a single 64-bit random value and get_rdrand64u_arr is used to return an array of 1000 64-bit random values.

```
//Check for RDRAND instruction support
int ret = is_RDRAND_supported();
int N = 1000;

//If RDRAND supported
if (ret == SECRNG_SUPPORTED)
{
  uint64_t rng64;

  //Get 64-bit random number
  ret = get_rdrand64u(&rng64, 0);

  if (ret == SECRNG_SUCCESS)
    printf("RDRAND rng 64-bit value %lu\n\n", rng64);
  else
    printf("Failure in retrieving random value using RDRAND!\n");

  //Get a range of 64-bit random values
  uint64_t* rng64_arr = (uint64_t*) malloc(sizeof(uint64_t) * N);

  ret = get_rdrand64u_arr(rng64_arr, N, 0);

  if (ret == SECRNG_SUCCESS)
  {
    printf("RDRAND for %u 64-bit random values succeeded!\n", N);
    printf("First 10 values in the range : \n");
    for (int i = 0; i < (N > 10? 10 : N); i++)
        printf("%lu\n", rng64_arr[i]);
  }
  else
    printf("Failure in retrieving array of random values using RDRAND!\n");
}
else
{
    printf("No support for RDRAND!\n");
}
```

7. FFTW

FFTW is a comprehensive collection of fast C routines for computing the Discrete Fourier Transform (DFT) and various special cases thereof. It is an open-source implementation of the Fast Fourier transform algorithm. It can compute transforms of real and complex-values arrays of arbitrary size and dimension.

7.1. Installation

FFTW can be installed either from source or pre-built binaries.

7.1.1. Build FFTW from source

Here are the steps to build FFTW for AMD EPYC processor

1. Download the latest stable release of FFTW 3.3.8 from the link
   https://github.com/FFTW/fftw3/archive/fftw-3.3.8.tar.gz
2. Untar the file
   $ tar xvzf fftw-3.3.8.tar.gz
   $ cd fftw-3.3.8
3. Depending on the target system, and build environment, one would have to enable/disable suitable configure options. The following steps provides instructions for compiling on AMD EPYC processors. For a complete list of options and their description, type ./configure –help

   $ ./configure --enable-sse2 --enable-fma --enable-avx --enable-avx2 --enable-avx-128-fma --enable-mpi --enable-shared --enable-openmp --prefix==*<your-install-dir>*
4. $ make
5. $ make install

7.1.2. Using pre-built binaries

FFTW binary can be installed from the tar file available in the following link. The tar file includes pre-built binaries of other AMD Libraries BLIS, libFLAME, LibM, Random Number Generator and AMD Secure RNG

https://developer.amd.com/amd-cpu-libraries/

7.2. Usage

Sample programs demonstrating usage FFTW APIs and performance benchmarking can be found under the tests directory of FFTW source

$ cd fftw-3.3.8/tests

8. Applications integrated to AMD CPU Libraries

This section provides examples on how AMD CPU Libraries can be linked with some of the important High Performance Computing (HPC) and cpu-intensive applications and libraries.

8.1. ScaLAPACK

ScaLAPACK[2] is a library of high-performance linear algebra routines for parallel distributed memory machines. It depends on external libraries including BLAS and LAPACK for Linear Algebra computations.

ScaLAPACK source directory includes a file called SLMake.inc, that is used to specify all the build configurations. All machine-specific parameters and the locations of BLAS and LAPACK libraries are specified in the SLmake.inc file.

Following is a sample SLMake.inc file with AMS BLIS and libFLAME libraries specified

```
############################################################################
#
#  Program:       ScaLAPACK
#
#  Module:        SLmake.inc
#
#  …………….
#
############################################################################
#
#  C preprocessor definitions:  set CDEFS to one of the following:
#
#     -DNoChange (fortran subprogram names are lower case without any suffix)
#     -DUpCase   (fortran subprogram names are upper case without any suffix)
#     -DAdd_     (fortran subprogram names are lower case with "_" appended)

CDEFS        = -DAdd_

#
#  The fortran and C compilers, loaders, and their flags
#

FC          = mpif90
CC          = mpicc
NOOPT        = -O0
FCFLAGS      = -O3
CCFLAGS      = -O3
FCLOADER     = $(FC)
CCLOADER     = $(CC)
FCLOADFLAGS  = $(FCFLAGS)
CCLOADFLAGS  = $(CCFLAGS)

#
#  The archiver and the flag(s) to use when building archive (library)
#  Also the ranlib routine.  If your system has no ranlib, set RANLIB = echo
#

ARCH         = ar
ARCHFLAGS    = cr
RANLIB       = ranlib
```

```
#
#  The name of the ScaLAPACK library to be created
#

SCALAPACKLIB  = libscalapack.a

#
#  BLAS, LAPACK (and possibly other) libraries needed for linking test programs
#

BLASLIB      = -L/home/pradeep/amd/libs/1.0/libs/ -lblis-mt
LAPACKLIB    = -L/home/pradeep/amd/libs/1.0/libs/ -lflame
LIBS         = $(LAPACKLIB) $(BLASLIB)
```

8.2. High-Performance LINPACK Benchmark (HPL)

HPL[3] is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. HPL is a LINPACK benchmark which measures the floating point rate of execution for solving a linear system of equations.

The following document provides details on the system configuration details and steps to install and run HPL on AMD EPYC CPU. https://developer.amd.com/wp-content/resources/56420.pdf

Additionally, a utility has been created that builds and runs HPL. It also builds AMD BLIS library that is used to link to HPL as the dependent BLAS library.

The utility can be downloaded here
https://github.com/amd/blis/releases/download/1.2/install_run_hpl-blis.tar.gz

The utility package contains following files
1. README.txt
2. build_run_hpl_blis.sh - Script to build and run HPL. Also pulls AMD BLIS library from github and builds and links to HPL
3. HPL.dat - Sample HPL dat file
4. Make.Linux_AMD_BLIS - Configuration file with AMD BLIS library paths

9. AMD CPU Libraries Tuning Guidelines

This section provides tuning recommendations for AMD CPU Libraries so as to derive best optimal performance on AMD EPYC™ and future generation architectures.

9.1. BLIS DGEMM multi-thread tuning

To achieve best DGEMM performance on AMD EPYC "Zen" core based processors, follow the below steps.

The header file, *bli_family_zen.h* located under BLIS source directory *\\blis\config\zen* defines certain macros that help control block sizes used by BLIS. Enabling and disabling these macros causes choosing the appropriate block sizes that BLIS operates on.

The required tuning settings vary depending on the number threads that the application linked to BLIS runs.

Thread Size upto 16 (< 16)

1. Enable the macro BLIS_ENABLE_ZEN_BLOCK_SIZES in the file bli_family_zen.h
2. Compile BLIS with multithread option as mention in section Multi-threaded BLIS
3. Link generated BLIS library to your application and execute

Thread Size above 16 (>= 16)

1. Disable the macro BLIS_ENABLE_ZEN_BLOCK_SIZES in the file bli_family_zen.h
2. Compile BLIS with multithread option as mention in section Multi-threaded BLIS
3. Link generated BLIS library to your application
4. Set the following OpenMP and memory interleaving environment settings
   OMP_PROC_BIND=spread
   OMP_NUM_THREADS = x   // x> 16
   numactl --interleave=all
5. Run the application
   Example:
   OMP_PROC_BIND=spread OMP_NUM_THREADS=32 numactl --interleave=all ./test_gemm_blis.x

9.2. BLIS DGEMM block size tuning for single and multi-instance mode

BLIS DGEMM performance is largely impacted by the block sizes used by BLIS. A matrix multiplication of large m, n and k dimensions is partitioned into sub-problems of specified block sizes[4].

Many high-performance computing (HPC) and scientific applications and benchmarks run on high end cluster of machines, each with multiple cores. They run programs with multiple instances which could be through Message Passing Interface (MPI) based APIs or separate instances of each program. Depending on whether the application using BLIS is running in multi-instance mode or single instance, the block sizes specified would have an impact on the overall performance.

The default values for the block size under AMD BLIS github repo is set to extract best performance for such HPC applications/benchmarks which use single-threaded BLIS and run in multi-instance mode on AMD EPYC "Zen" core processors. However, if your application runs as a single instance, the block sizes for optimal performance will vary.

Following settings will help you choose the optimal values for the block sizes based on the way application is run

1. Open the file *bli_cntx_init_zen.c* under BLIS source
   $ cd *"config/zen/bli_family_zen.h"*

2. Ensure the macro, BLIS_ENABLE_ZEN_BLOCK_SIZES is defined

#define BLIS_ENABLE_ZEN_BLOCK_SIZES

3. **Multi-instance mode**:
   For applications/benchmarks running in multi-instance mode, ensure the macro BLIS_ENABLE_SINGLE_INSTANCE_BLOCK_SIZES is set to 0. As of AMD BLIS 1.3 release, this is the default setting

#define BLIS_ENABLE_SINGLE_INSTANCE_BLOCK_SIZES   0

   The optimal block sizes for this mode on AMD EPYC are defined in the file *"config/zen/bli_cntx_init_zen.c"*

   ```
   bli_blksz_init_easy( &blkszs[ BLIS_MC ],  144, 240,  144,   72 );
   bli_blksz_init_easy( &blkszs[ BLIS_KC ],  256, 512,  256,  256 );
   bli_blksz_init_easy( &blkszs[ BLIS_NC ], 4080, 2040, 4080, 4080 );
   ```

4. **Single instance mode**:
   For applications running as a single instance, ensure the macro BLIS_ENABLE_SINGLE_INSTANCE_BLOCK_SIZES is set to 1.

#define BLIS_ENABLE_SINGLE_INSTANCE_BLOCK_SIZES   1

   The optimal block sizes for this mode on AMD EPYC are defined in the file *"config/zen/bli_cntx_init_zen.c"*

   ```
   bli_blksz_init_easy( &blkszs[ BLIS_MC ],  144, 510,  144,   72 );
   bli_blksz_init_easy( &blkszs[ BLIS_KC ],  256, 1024, 256,  256 );
   bli_blksz_init_easy( &blkszs[ BLIS_NC ], 4080, 4080, 4080, 4080 );
   ```

10. Technical Support and Forums

AMD Libraries Forum can be used for questions and issues about AMD CPU Libraries

https://community.amd.com/community/devgurus/amd-compute-libraries

Also, one can reach us on the following email-id cpulibsupport@amd.com

11.   References

1. https://developer.amd.com/amd-cpu-libraries/
2. http://www.netlib.org/scalapack/
3. http://www.netlib.org/benchmark/hpl/
4. https://dl.acm.org/citation.cfm?id=2764454

DISCLAIMER
The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.
AMD, the AMD Arrow logo, EPYC and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.