# Intel® oneAPI Math Kernel Library for macOS*

**Developer Guide**

***Intel® oneAPI Math Kernel Library- macOS****

Notices and Disclaimers

***2023.0***

# Contents

# Notices and Disclaimers

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.<br><br>Notice revision #20201201 |

# *Getting Help and Support*

Intel provides a support web site that contains a rich repository of self help information, including getting started tips, known product issues, product errata, license information, user forums, and more. Visit the Intel® oneAPI Math Kernel Library support website athttp://www.intel.com/software/products/support/.

# Introducing the Intel® oneAPI Math Kernel Library

Intel® oneAPI Math Kernel Library (oneMKL) is a computing math library of highly optimized, extensively threaded routines for applications that require maximum performance. The library provides Fortran and C programming language interfaces. oneMKL C language interfaces can be called from applications written in either C or C++, as well as in any other language that can reference a C interface.

oneMKL provides comprehensive functionality support in these major areas of computation:

- BLAS (level 1, 2, and 3) and LAPACK linear algebra routines, offering vector, vector-matrix, and matrix-matrix operations.
- ScaLAPACK distributed processing linear algebra routines, as well as the Basic Linear Algebra Communications Subprograms (BLACS) and the Parallel Basic Linear Algebra Subprograms (PBLAS).
- oneMKL PARDISO (a direct sparse solver based on Parallel Direct Sparse Solver PARDISO*), an iterative sparse solver, and supporting sparse BLAS (level 1, 2, and 3) routines for solving sparse systems of equations, as well as a distributed version of oneMKL PARDISO solver provided for use on clusters.
- Fast Fourier transform (FFT) functions in one, two, or three dimensions with support for mixed radices (not limited to sizes that are powers of 2), as well as distributed versions of these functions provided for use on clusters.
- Vector Mathematics (VM) routines for optimized mathematical operations on vectors.
- Vector Statistics (VS) routines, which offer high-performance vectorized random number generators (RNG) for several probability distributions, convolution and correlation routines, and summary statistics functions.
- Data Fitting Library, which provides capabilities for spline-based approximation of functions, derivatives and integrals of functions, and search.
- Extended Eigensolver, a shared memory programming (SMP) version of an eigensolver based on the Feast Eigenvalue Solver.

For details see the *Intel® oneAPI Math Kernel Library Developer Reference*.

Intel® oneAPI Math Kernel Library (oneMKL) is optimized for performance on Intel processors. oneMKL also runs on non-Intel x86-compatible processors.

**NOTE**
oneMKL provides limited input validation to minimize the performance overheads. It is your responsibility when using oneMKL to ensure that input data has the required format and does not contain invalid characters. These can cause unexpected behavior of the library. Examples of the inputs that may result in unexpected behavior:

- Not-a-number (NaN) and other special floating point values
- Large inputs may lead to accumulator overflow

As the oneMKL API accepts raw pointers, it is your application's responsibility to validate the buffer sizes before passing them to the library. The library requires subroutine and function parameters to be valid before being passed. While some oneMKL routines do limited checking of parameter errors, your application should check for NULL pointers, for example.

# Notational Conventions

The following term is used in reference to the operating system.

| | |
|---|---|
| macOS* | This term refers to information that is valid on all supported macOS* and OS X* operating systems. |

The following notations are used to refer to Intel® oneAPI Math Kernel Library directories.

| | |
|---|---|
| *`<parent directory>`* | The installation directory that includes Intel® oneAPI Math Kernel Library directory; for example, the directory for Intel® Parallel Studio XE Composer Edition. |
| *`<mkl directory>`* | The main directory where Intel® oneAPI Math Kernel Library is installed: *`<mkl directory>`*`=`*`<parent directory>`*`/mkl`. Replace this placeholder with the specific pathname in the configuring, linking, and building instructions. |

The following font conventions are used in this document.

| | |
|---|---|
| *Italic* | Italic is used for emphasis and also indicates document names in body text, for example: see *Intel® oneAPI Math Kernel Library Developer Reference*. |
| `Monospace lowercase mixed with uppercase` | Indicates: • Commands and command-line options, for example, `icc myprog.c -L$MKLPATH -I$MKLINCLUDE -lmkl -liomp5 -lpthread` • Filenames, directory names, and pathnames, for example, `/System/Library/Frameworks/JavaVM.framework/Versions/1.5/Home` • C/C++ code fragments, for example, `a = new double [SIZE*SIZE];` |
| `UPPERCASE MONOSPACE` | Indicates system variables, for example, `$MKLPATH`. |
| *`Monospace italic`* | Indicates a parameter in discussions, for example, *`lda`*. When enclosed in angle brackets, indicates a placeholder for an identifier, an expression, a string, a symbol, or a value, for example, *`<mkl directory>`*. Substitute one of these items for the placeholder. |
| `[ items ]` | Square brackets indicate that the items enclosed in brackets are optional. |
| `{ item \| item }` | Braces indicate that only one of the items listed between braces should be selected. A vertical bar ( \| ) separates the items. |

# *Related Information*

To reference how to use the library in your application, use this guide in conjunction with the following documents:

- The *Intel® oneAPI Math Kernel Library Developer Reference*, which provides *reference* information on routine functionalities, parameter descriptions, interfaces, calling syntaxes, and return values.
- The *Intel® oneAPI Math Kernel Library for OS X* Release Notes*.

Web versions of these documents are available in the Intel® Software Documentation Library at https://software.intel.com/content/www/us/en/develop/documentation.html.

# Getting Started

1

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201

## Shared Library Versioning

Intel® oneAPI Math Kernel Library (oneMKL) adds shared library versioning for all operating systems and platforms, as opposed to not using any library versioning up to Intel® Math Kernel Library (Intel® MKL) 2020 Update 4.

This new feature:

- Allows applications to work correctly in an environment with multiple oneMKL and/or Intel® MKL packages installed
- Communicates clearly when incompatible changes are made, and an application should be rebuilt
- Allows you to link against a specific version of shared libraries

The starting version for shared libraries is "1" and any change that break backward compatibility will result in an increment to this number. Intel expects to make this change as seldom as possible and inform customers about it at least 24 months in advance.

The product version "2021.1" is now decoupled from the library version, meaning that "2021.2" can ship with shared libraries versioned as "1". This means that the libraries shipped in "2021.2" are backward compatible with libraries shipped in "2021.1".

Changes to the link-line:

- No changes are required to the link-line because symbolic links are provided with the old names, which point to the new library that contains the version information on Linux* and MacOS*. The symbolic link name is also the `soname` and `install_name` of that library on Linux and MacOS, respectively.
  - For example, `libmkl_core.so` -> `libmkl_core.<version>.so`
  - For example, `libmkl_core.dylib` -> `libmkl_core.<version>.dylib`
  - Using `-lmkl_core` will still work as before, ensuring backward compatibility with Intel® MKL 2020 line-up (including Intel® Parallel Studio and Intel® System Studio distributions).
- On Windows*, import libraries used in the link-line do not contain any version information, as before, but point to the new DLL, which contains the version information.
  - For example, `mkl_core_dll.lib` has the same name as before and requires no change to the link-line. The linker, however, resolves this to the new `mkl_core.<version>.dll` instead of the older `mkl_core.dll`.

## CMake Config for oneMKL

If you want to integrate oneMKL into your CMake projects, starting with the Intel® oneAPI Math Kernel Library (oneMKL) 2021.3 release, `MKLConfig.cmake` is provided as part of the package and installation. `MKLConfig.cmake` supports all oneMKL configurations, compilers, and runtimes, as the oneMKL product itself. Help/usage is provided in the top section of `MKLConfig.cmake`.

### Example

```
my_test/
  |_ build/               <-- Out-of-source build directory
  |_ CMakeLists.txt       <-- User side project's CMakeLists.txt
  |_ app.c                <-- Source file that uses oneMKL API
```

**CMakeLists.txt**

```
cmake_minimum_required(VERSION 3.13)
enable_testing()
project(oneMKL_Example LANGUAGES C)
find_package(MKL CONFIG REQUIRED)
#message(STATUS "${MKL_IMPORTED_TARGETS}") #Provides available list of targets based on input
add_executable(myapp app.c)

target_compile_options(myapp PUBLIC $<TARGET_PROPERTY:MKL::MKL,INTERFACE_COMPILE_OPTIONS>)
target_include_directories(myapp PUBLIC
$<TARGET_PROPERTY:MKL::MKL,INTERFACE_INCLUDE_DIRECTORIES>)
target_link_libraries(myapp PUBLIC $<LINK_ONLY:MKL::MKL>)

add_test(NAME mytest COMMAND myapp)
if(MKL_ENV)
  set_tests_properties(mytest PROPERTIES ENVIRONMENT "${MKL_ENV}")
endif()
```

**Command line**

```
# Source the compiler and runtime beforehand
build$ cmake .. -DCMAKE_C_COMPILER=icc
build$ cmake --build . && ctest
# If MKLConfig.cmake is not located by CMake automatically, its path can be manually specified
by MKL_DIR:
build$ cmake .. -DCMAKE_C_COMPILER=icc -DMKL_DIR=<Full path to MKLConfig.cmake>
```

**NOTE**
When the Ninja build system is in use, Ninja 1.10.2+ is required for Fortran support.

## Checking Your Installation

After installing the Intel® oneAPI Math Kernel Library (oneMKL), verify that the library is properly installed and configured:

1. Intel® oneAPI Math Kernel Library installs in the`<parent directory>` directory.

   Check that the subdirectory of `<parent directory>` referred to as `<mkl directory>` was created.
2. If you want to keep multiple versions of Intel® oneAPI Math Kernel Library installed on your system, update your build scripts to point to the correct Intel® oneAPI Math Kernel Library version.
3. Check that the following files appear in the `<mkl directory>/env` directory:

   `vars.sh`

   Use these files to assign Intel® oneAPI Math Kernel Library-specific values to several environment variables, as explained in Scripts to Set Environment Variables Setting Environment Variables .
4. To understand how the Intel® oneAPI Math Kernel Library directories are structured, seeStructure of the Intel® Math Kernel Library.
5. To make sure that Intel® oneAPI Math Kernel Library runs on your system, launch an Intel® oneAPI Math Kernel Library example, as explained inUsing Code Examples.

### See Also

Notational Conventions

## Setting Environment Variables

When the installation of Intel® oneAPI Math Kernel Library for macOS* is complete, set environment variables `INCLUDE`, `MKLROOT`, `LD_LIBRARY_PATH`, `LIBRARY_PATH`, `CPATH`, `NLSPATH`, and `PKG_CONFIG_PATH` using the setvars script.

The scripts accept the oneMKL-specific parameters, explained in the following table:

| **Setting Specified** | **Required (Yes/No)** | **Possible Values** | **Comment** |
|---|---|---|---|
| Architecture | Yes, when applicable | `intel64` | |
| Use of Intel® oneAPI Math Kernel Library Fortran modules precompiled with the Intel®Fortran compiler | No | `mod` | Supply this parameter only if you are using this compiler. |
| Programming interface (LP64 or ILP64) | No | `lp64`, default<br>`ilp64` | |

For example:

- The command `setvars.sh intel64 mod ilp64`
  sets the environment for Intel® oneAPI Math Kernel Library to use the Intel 64 architecture, ILP64 programming interface, and Fotran modules.
- The command `setvars.sh intel64 mod`
  sets the environment for Intel® oneAPI Math Kernel Library to use the Intel 64 architecture, LP64 interface, and Fotran modules.

**NOTE**
Supply the parameter specifying the architecture first, if it is needed. Values of the other two parameters can be listed in any order.

### See Also
High-level Directory Structure
Intel® oneAPI Math Kernel Library Interface Libraries and Modules
Fortran 95 Interfaces to LAPACK and BLAS
Setting the Number of Threads Using an OpenMP* Environment Variable

## Compiler Support

When building Intel® oneAPI Math Kernel Library code examples, you can select a compiler:

- For Fortran examples: Intel® compiler
- For C examples: Intel, Clang*, or GNU* compiler

Intel® oneAPI Math Kernel Library provides a set of include files to simplify program development by specifying enumerated values and prototypes for the respective functions. Calling Intel® oneAPI Math Kernel Library functions from your application without an appropriate include file may lead to incorrect behavior of the functions.

### See Also
Intel® oneAPI Math Kernel Library Include Files

## Using Code Examples

The Intel® oneAPI Math Kernel Library package includes code examples, located in the`examples` subdirectory of the installation directory. Use the examples to determine:

- Whether Intel® oneAPI Math Kernel Library is working on your system
- How you should call the library
- How to link the library

If an Intel® oneAPI Math Kernel Library component that you selected during installation includes code examples, these examples are provided in a separate archive. Extract the examples from the archives before use.

For each component, the examples are grouped in subdirectories mainly by Intel® oneAPI Math Kernel Library function domains and programming languages. For instance, the`blas` subdirectory (extracted from the `examples_core` archive) contains a makefile to build the BLAS examples and the `vmlc` subdirectory contains the makefile to build the C examples for Vector Mathematics functions. Source code for the examples is in the next-level `sources` subdirectory.

### See Also

High-level Directory Structure

## What You Need to Know Before You Begin Using the Intel® oneAPI Math Kernel Library

Target platform

Identify the architecture of your target machine:

- 
- Intel® 64 or compatible

**Reason:** Linking ExamplesTo configure your development environment for the use with Intel® oneAPI Math Kernel Library, set your environment variables using the script corresponding to your architecture (see Scripts to Set Environment Variables Setting Environment Variables for details).

Mathematical problem

Identify all Intel® oneAPI Math Kernel Library function domains that you require:

- BLAS
- Sparse BLAS
- LAPACK
- PBLAS
- ScaLAPACK
- Sparse Solver routines
- Parallel Direct Sparse Solvers for Clusters
- Vector Mathematics functions (VM)
- Vector Statistics functions (VS)
- Fourier Transform functions (FFT)
- Cluster FFT
- Trigonometric Transform routines
- Poisson, Laplace, and Helmholtz Solver routines
- Optimization (Trust-Region) Solver routines
- Data Fitting Functions
- Extended Eigensolver Functions

**Reason:** The function domain you intend to use narrows the search in the *Intel® oneAPI Math Kernel Library Developer Reference*for specific routines you need. Additionally, if you are using the Intel® oneAPI Math Kernel Library cluster software, your link line is function-domain specific (seeWorking with the Intel® oneAPI Math Kernel Library Cluster Software). Coding tips may also depend on the function domain (see Other Tips and Techniques to Improve Performance).

Programming language

Intel® oneAPI Math Kernel Library provides support for both Fortran and C/C++ programming. Identify the language interfaces that your function domains support (see Appendix A: Intel® oneAPI Math Kernel Library Language Interfaces Support).

**Reason:**Intel® oneAPI Math Kernel Library provides language-specific include files for each function domain to simplify program development (seeLanguage Interfaces Support_ by Function Domain).

For a list of language-specific interface libraries and modules and an example how to generate them, see also Using Language-Specific Interfaces with Intel® oneAPI Math Kernel Library.

Range of integer data

If your system is based on the Intel 64 architecture, identify whether your application performs calculations with large data arrays (of more than $2^{31}$-1 elements).

**Reason:** To operate on large data arrays, you need to select the ILP64 interface, where integers are 64-bit; otherwise, use the default, LP64, interface, where integers are 32-bit (see Using the ILP64 Interface vs).

Threading model

Identify whether and how your application is threaded:

- Threaded with the Intel compiler
- Threaded with a third-party compiler
- Not threaded

**Reason:**The compiler you use to thread your application determines which threading library you should link with your application. For applications threaded with a third-party compiler you may need to use Intel® oneAPI Math Kernel Library in the sequential mode (for more information, seeLinking with Threading Libraries).

Number of threads

If your application uses an OpenMP* threading run-time library, determine the number of threads you want Intel® oneAPI Math Kernel Library to use.

**Reason:**By default, the OpenMP* run-time library sets the number of threads for Intel® oneAPI Math Kernel Library. If you need a different number, you have to set it yourself using one of the available mechanisms. For more information, seeImproving Performance with Threading.

Linking model

Decide which linking model is appropriate for linking your application with Intel® oneAPI Math Kernel Library libraries:

- Static
- Dynamic

**Reason:** The link line syntax and libraries for static and dynamic linking are different. For the list of link libraries for static and dynamic models, linking examples, and other relevant topics, like how to save disk space by creating a custom dynamic library, see Linking Your Application with the Intel® oneAPI Math Kernel Library.

# Structure of the Intel® oneAPI Math Kernel Library

2

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201

## Architecture Support

Intel® oneAPI Math Kernel Library (oneMKL) for macOS*supports the Intel® 64 architecture, located in the`<mkl directory>/lib` directory.

**See Also**

High-level Directory Structure

Notational Conventions

Directory Structure in Detail

## High-level Directory Structure

| Directory | Contents |
|---|---|
| `<mkl directory>` | Installation directory of the Intel® oneAPI Math Kernel Library (oneMKL) |
| **Subdirectories of**`<mkl directory>` | |
| `bin/` | Scripts to set environmental variables in the user shell |
| `bin/intel64` | Shell scripts for the Intel® 64 architecture |
| `benchmarks/linpack` | Shared-Memory (SMP) version of LINPACK benchmark |
| `examples` | Source and data files for Intel® oneAPI Math Kernel Library examples. Provided in archives corresponding to Intel® oneAPI Math Kernel Library components selected during installation. |
| `include` | Include files for the library routines and examples |
| `include/intel64/lp64` | Fortran 95 .mod files for the Intel® 64 architecture, Intel Fortran compiler, and LP64 interface |
| `include/intel64/ilp64` | Fortran 95 .mod files for the Intel® 64 architecture, Intel® Fortran compiler, and ILP64 interface |
| `include/fftw` | Header files for the FFTW2 and FFTW3 interfaces |
| `interfaces/blas95` | Fortran 95 interfaces to BLAS and a makefile to build the library |
| `interfaces/fftw2xc` | FFTW 2.x interfaces to Intel® oneAPI Math Kernel Library FFT (C interface) |
| `interfaces/fftw2xf` | FFTW 2.x interfaces to Intel® oneAPI Math Kernel Library FFT (Fortran interface) |

| Directory | Contents |
|---|---|
| `interfaces/fftw3xc` | FFTW 3.x interfaces to Intel® oneAPI Math Kernel Library FFT (C interface) |
| `interfaces/fftw3xf` | FFTW 3.x interfaces to Intel® oneAPI Math Kernel Library FFT (Fortran interface) |
| `interfaces/lapack95` | Fortran 95 interfaces to LAPACK and a makefile to build the library |
| `lib` | Universal static libraries and shared objects for the Intel® 64 architecture |
| `tools` | Tools and plug-ins |
| `tools/builder` | Tools for creating custom dynamically linkable libraries |

**See Also**
Notational Conventions
Using Code Examples

## Layered Model Concept

Intel® oneAPI Math Kernel Library is structured to support multiple compilers and interfaces, both serial and multi-threaded modes, different implementations of threading run-time libraries, and a wide range of processors. Conceptually Intel® oneAPI Math Kernel Library can be divided into distinct parts to support different interfaces, threading models, and core computations:

1. Interface Layer
2. Threading Layer
3. Computational Layer

You can combine Intel® oneAPI Math Kernel Library libraries to meet your needs by linking with one library in each part layer-by-layer.

To support threading with different compilers, you also need to use an appropriate threading run-time library (RTL). These libraries are provided by compilers and are not included in Intel® oneAPI Math Kernel Library.

The following table provides more details of each layer.

| Layer | Description |
|---|---|
| Interface Layer | This layer matches compiled code of your application with the threading and/or computational parts of the library. This layer provides: • LP64 and ILP64 interfaces. • Compatibility with compilers that return function values differently. |
| Threading Layer | This layer: • Provides a way to link threaded Intel® oneAPI Math Kernel Library with supported compilers. • Enables you to link with a threaded or sequential mode of the library. This layer is compiled for different environments (threaded or sequential) and compilers (from Intel ). |
| Computational Layer | This layer accommodates multiple architectures through identification of architecture features and chooses the appropriate binary code at run time. |

**See Also**
Using the ILP64 Interface vs. LP64 Interface
Linking Your Application with the Intel® oneAPI Math Kernel Library

## Linking with Threading Libraries

# Linking Your Application with the Intel® oneAPI Math Kernel Library

3

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.<br>Notice revision #20201201 |

## Linking Quick Start

Intel® oneAPI Math Kernel Library (oneMKL) provides several options for quick linking of your application, which depend on the way you link:

| | |
|---|---|
| Using the Intel® Parallel Studio XE Composer Edition compiler | see Using the -mkl Compiler Option. |
| Explicit dynamic linking | see Using the Single Dynamic Library for how to simplify your link line. |
| Explicitly listing libraries on your link line | see Selecting Libraries to Link with for a summary of the libraries. |
| Using pkg-config tool to get compilation and link lines | see Using pkg-config metadata filesfor a summary on how to use Intel® oneAPI Math Kernel Library pkg-config metadata files. |
| Using an interactive interface | see Using the Link-line Advisor to determine libraries and options to specify on your link or compilation line. |
| Using an internally provided tool | see Using the Command-line Link Tool to determine libraries, options, and environment variables or even compile and build your application. |

### Using the -qmkl Compiler Option

The Intel®Parallel Studio XE Composer Edition compiler supports the following variants of the `-qmkl` compiler option:

| | |
|---|---|
| `-qmkl` | to link with a certain Intel® oneAPI Math Kernel Library threading layer depending on the threading option provided:<br>• For `-qopenmp` the OpenMP threading layer for Intel compilers<br>• For `-tbb` the Intel® Threading Building Blocks (Intel® TBB) threading layer |
| `-qmkl=parallel` | to link with a certain Intel® oneAPI Math Kernel Library threading layer depending on the threading option provided:<br>• For `-qopenmp` the OpenMP threading layer for Intel compilers |

| | |
|---|---|
| | • For `-tbb` the Intel® Threading Building Blocks (Intel® TBB) threading layer |
| `-qmkl=sequential` | to link with sequential version of Intel® oneAPI Math Kernel Library. |
| `-qmkl=cluster` | to link with Intel® oneAPI Math Kernel Library cluster components (sequential) that use Intel MPI. |

**NOTE**
The `-qopenmp` option has higher priority than `-tbb`in choosing the Intel® oneAPI Math Kernel Library threading layer for linking.

For more information on the `-qmkl` compiler option, see the Intel Compiler User and Reference Guides.

On Intel® 64 architecture systems, for each variant of the `-qmkl` option, the compiler links your application using the LP64 interface.

If you specify any variant of the `-qmkl` compiler option, the compiler automatically includes the Intel® oneAPI Math Kernel Library libraries. In cases not covered by the option, use the Link-line Advisor or see Linking in Detail.

### See Also

Listing Libraries on a Link Line
Intel® oneAPI DPC++/C++ Compiler Developer Guide and Reference

Intel® Fortran Compiler Classic and Intel® Fortran Compiler Developer Guide and Reference

Using the ILP64 Interface vs. LP64 Interface
Using the Link-line Advisor
Intel® Software Documentation Library for Intel® compiler documentation
for Intel® compiler documentation

## Using the Single Dynamic Library

You can simplify your link line through the use of the Intel® oneAPI Math Kernel Library Single Dynamic Library (SDL).

To use SDL, place `libmkl_rt.dylib` on your link line. For example:

```
icc application.c -L$MKLPATH -Wl,-rpath,$MKLPATH -lmkl_rt
```

Here `MKLPATH=$MKLROOT/lib`.

SDL enables you to select the interface and threading library for Intel® oneAPI Math Kernel Library at run time. By default, linking with SDL provides:

- Intel LP64 interface on systems based on the Intel® 64 architecture
- Intel threading

To use other interfaces or change threading preferences, including use of the sequential version of Intel® oneAPI Math Kernel Library, you need to specify your choices using functions or environment variables as explained in sectionDynamically Selecting the Interface and Threading Layer.

**NOTE**Intel® oneAPI Math Kernel Library SDL (mkl_rt) does not support DPC++ APIs. If your application requires support of Intel® oneAPI Math Kernel Library DPC++ APIs, refer to Intel® oneAPI Math Kernel Library Link-line Advisor to configure your link command.

## Selecting Libraries to Link with

To link with Intel® oneAPI Math Kernel Library:

- Choose one library from the Interface layer and one library from the Threading layer
- Add the only library from the Computational layer and run-time libraries (RTL)

The following table lists Intel® oneAPI Math Kernel Library libraries to link with your application.

| | **Interface layer** | **Threading layer** | **Computational layer** | **RTL** |
|---|---|---|---|---|
| **Intel® 64 architecture, static linking** | `libmkl_intel_lp64.a` | `libmkl_intel_thread.a` | `libmkl_core.a` | `libiomp5.dylib` |
| **Intel® 64 architecture, dynamic linking** | `libmkl_intel_lp64. dylib` | `libmkl_intel_thread.dylib` | `libmkl_core.dylib` | `libiomp5.dylib` |

The Single Dynamic Library (SDL) automatically links interface, threading, and computational libraries and thus simplifies linking. The following table lists Intel® oneAPI Math Kernel Library libraries for dynamic linking using SDL. SeeDynamically Selecting the Interface and Threading Layer for how to set the interface and threading layers at run time through function calls or environment settings.

| | **SDL** | **RTL** |
|---|---|---|
| **Intel® 64 architecture** | `libmkl_rt.dylib` | `libiomp5.dylib`† |

†Use the Link-line Advisor to check whether you need to explicitly link the `libiomp5.dylib` RTL.

For exceptions and alternatives to the libraries listed above, see Linking in Detail.

### See Also

Layered Model Concept
Using the Link-line Advisor
Using the `-mkl` Compiler Option

## Using the Link-line Advisor

Use the Intel® oneAPI Math Kernel Library Link-line Advisor to determine the libraries and options to specify on your link or compilation line.

The latest version of the tool is available at Link Line Advisor for Intel® oneAPI Math Kernel Library. The tool is also available in the documentation directory of the product.

The Advisor requests information about your system and on how you intend to use Intel® oneAPI Math Kernel Library (link dynamically or statically, use threaded or sequential mode, and so on). The tool automatically generates the appropriate link line for your application.

### See Also

High-level Directory Structure

## Using the Command-Line Link Tool

Use the command-line Link Tool provided by Intel® oneAPI Math Kernel Library to simplify building your application with Intel® oneAPI Math Kernel Library.

The tool not only provides the options, libraries, and environment variables to use, but also performs compilation and building of your application.

The tool `mkl_link_tool` is installed in the `<mkl_directory>/bin/<arch>` directory, and supports the modes described in the following table.

**oneMKL Command-Line Link Tool Modes**

| Mode | Description | Usage | Example |
|---|---|---|---|
| Inquiry | The tool returns the compiler options, libraries, or environment variables necessary to build and execute the application. | Get Intel® oneAPI Math Kernel Library libraries | `mkl_link_tool -libs [oneMKL Link Tool options]` |
| | | Get compilation options | `mkl_link_tool -opts [oneMKL Link Tool options]` |
| | | Get environment variables for application executable | `mkl_link_tool -env [oneMKL Link Tool options]` |
| Compilation | The Intel® oneAPI Math Kernel Library Link Tool builds the application. | — | `mkl_link_tool [options] <compiler> [options2] file1 [file2 ...]`<br>where:<br>• `options` represents any number of Link Tool options<br>• `compiler` represents the compiler name: ifort, icc (icpc, icl), cl, gcc (g++), gfortran, pgcc (pgcp), pgf77 (pgf90, pgf95, pgfortran), mpiic, mpiifort, mpic (mpic++), mpif77 (mpif90, mpif95), icpx -fsycl<br>• `options2` represents any number of compiler options |
| Interactive | Allows you to go through all possible Intel® oneAPI Math Kernel Library Link Tool supported options. The output provides libraries, options, or environment variables as in the inquiry mode, or a built application as in the compilation mode (depending on what you specify). | — | `mkl_link_tool -interactive` |

Use the `-help` option for full help with the Intel® oneAPI Math Kernel Library Link Tool and to show the defaults for the current system.

## Linking Examples

### See Also
Using the Link-line Advisor

### Linking on Intel(R) 64 Architecture Systems

The following examples illustrate linking that uses Intel(R) compilers.

Most examples use the `.f` Fortran source file. C/C++ users should instead specify a `.cpp` (C++) or `.c` (C) file and replace `ifort` with `icc`.

In these examples,
`MKLPATH=$MKLROOT/lib`,
`MKLINCLUDE=$MKLROOT/include`.

**NOTE**
If you successfully completed the Scripts to Set Environment Variables Setting Environment Variables step of the Getting Started process, you can omit `-I$MKLINCLUDE` in all the examples and omit `-L$MKLPATH` in the examples for dynamic linking.

- Static linking of `myprog.f` and OpenMP* threadedIntel® oneAPI Math Kernel Library supporting the LP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
  $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
  $MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
  $MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
  ```
- Dynamic linking of `myprog.f` and OpenMP* threadedIntel® oneAPI Math Kernel Library supporting the LP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -Wl,-rpath,$MKLPATH
  -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
  -liomp5 -lpthread -lm
  ```
- Static linking of `myprog.f` and sequential version of Intel® oneAPI Math Kernel Library supporting the LP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
  $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_sequential.a
  $MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_sequential.a
  $MKLPATH/libmkl_core.a -lpthread -lm
  ```
- Dynamic linking of `myprog.f` and sequential version of Intel® oneAPI Math Kernel Library supporting the LP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -Wl,-rpath,$MKLPATH
  -lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread -lm
  ```
- Static linking of `myprog.f` and OpenMP* threadedIntel® oneAPI Math Kernel Library supporting the ILP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
  $MKLPATH/libmkl_intel_ilp64.a $MKLPATH/libmkl_intel_thread.a
  $MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_ilp64.a $MKLPATH/libmkl_intel_thread.a
  $MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
  ```
- Dynamic linking of `myprog.f` and OpenMP* threadedIntel® oneAPI Math Kernel Library supporting the ILP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -Wl,-rpath,$MKLPATH
  -lmkl_intel_ilp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm
  ```

- Dynamic linking of user code `myprog.f` and OpenMP* threadedor sequential Intel® oneAPI Math Kernel Library(Call appropriate functions or set environment variables to choose threaded or sequential mode and to set the interface):

  ```
  ifort myprog.f -L$MKLPATH -Wl,-rpath,$MKLPATH -lmkl_rt
  ```
- Static linking of `myprog.f`, Fortran BLAS and 95 LAPACK interfaces, and OpenMP* threadedIntel® oneAPI Math Kernel Library supporting the LP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/intel64/lp64
  -lmkl_lapack95_lp64 $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
  $MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
  $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
  ```
- Static linking of `myprog.c` and Intel® oneAPI Math Kernel Library threaded with Intel® Threading Building Blocks (Intel® TBB), provided that the `LIBRARY_PATH` environment variable contains the path to Intel TBB library:

  ```
  icc myprog.c -L$MKLPATH -I$MKLINCLUDE -Wl,--start-group $MKLPATH/libmkl_intel_lp64.a
  $MKLPATH/libmkl_tbb_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group -ltbb -lstdc++
  -lpthread -lm
  ```
- Dynamic linking of `myprog.c` and Intel® oneAPI Math Kernel Library threaded with Intel® TBB, provided that the `LIBRARY_PATH` environment variable contains the path to Intel® TBB library:

  ```
  icc myprog.c -L$MKLPATH -I$MKLINCLUDE -Wl,-rpath,$MKLPATH
  -lmkl_intel_lp64 -lmkl_tbb_thread -lmkl_core
  -ltbb -lstdc++ -lpthread -lm
  ```

**See Also**

Fortran 95 Interfaces to LAPACK and BLAS

Linking with System Libraries for specifics of linking with a GNU or PGI compiler

## Linking in Detail

This section recommends which libraries to link with depending on your Intel® oneAPI Math Kernel Library usage scenario and provides details of the linking.

### Listing Libraries on a Link Line

To link with Intel® oneAPI Math Kernel Library, specify paths and libraries on the link line as shown below.

> **NOTE**
> The syntax below is for dynamic linking. For static linking, replace each library name preceded with "-l" with the path to the library file. For example, replace `-lmkl_core` with `$MKLPATH/libmkl_core.a`, where `$MKLPATH` is the appropriate user-defined environment variable.

```
<files to link>

-L<MKL path>-I<MKL include>
[-I<MKL include>intel64|{ilp64|lp64}}]

[-Wl,-rpath,<MKL path>]

[-lmkl_blas{95|95_ilp64|95_lp64}]
[-lmkl_lapack{95|95_ilp64|95_lp64}]


-lmkl_{intel|intel_ilp64|intel_lp64}

-lmkl_{intel_thread|tbb_thread|sequential}
```

```
-lmkl_core

[-liomp5] [-lpthread] [-lm] [-ldl] [-ltbb -lstdc++]
```

In the case of dynamic linking, the `-rpath`linker option adds the location of Intel® oneAPI Math Kernel Library dynamic libraries to application run-path search paths for compliance with system integrity protection, introduced in OS X* 10.11. For more details, see:

- System Integrity Protection Guide (https://developer.apple.com/library/tvos/documentation/Security/Conceptual/System_Integrity_Protection_Guide/System_Integrity_Protection_Guide.pdf)
- Run-Path Dependent Libraries (https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/RunpathDependentLibraries.html)

In the case of static linking,for all components except BLAS and FFT, repeat interface, threading, and computational libraries two times (for example, `libmkl_intel_ilp64.a libmkl_intel_thread.a libmkl_core.a libmkl_intel_ilp64.a libmkl_intel_thread.a libmkl_core.a`). For the LAPACK component, repeat the threading and computational libraries three times.

The order of listing libraries on the link line is essential.

### See Also

Using the Link Line Advisor
Linking Examples

## Dynamically Selecting the Interface and Threading Layer

The Single Dynamic Library (SDL) enables you to dynamically select the interface and threading layer for Intel® oneAPI Math Kernel Library.

### Setting the Interface Layer

To set the interface layer at run time, use the `mkl_set_interface_layer` function or the `MKL_INTERFACE_LAYER` environment variable.

Available interface layers depend on the architecture of your system.

On systems based on the Intel® 64 architecture, LP64 and ILP64 interfaces are available. The following table provides values to be used to set each interface layer.

**Specifying the Interface Layer**

| Interface Layer | Value of **`MKL_INTERFACE_LAYER`** | Value of the Parameter of **`mkl_set_interface_layer`** |
|---|---|---|
| Intel LP64, default | `LP64` | `MKL_INTERFACE_LP64` |
| Intel ILP64 | `ILP64` | `MKL_INTERFACE_ILP64` |

If the `mkl_set_interface_layer` function is called, the environment variable `MKL_INTERFACE_LAYER` is ignored.

### Setting the Threading Layer

To set the threading layer at run time, use the `mkl_set_threading_layer` function or the `MKL_THREADING_LAYER` environment variable. The following table lists available threading layers along with the values to be used to set each layer.

### Specifying the Threading Layer

| Threading Layer | Value of `MKL_THREADING_LAYER` | Value of the Parameter of `mkl_set_threading_layer` |
|---|---|---|
| Intel threading, default | `INTEL` | `MKL_THREADING_INTEL` |
| Sequential mode of Intel® oneAPI Math Kernel Library | `SEQUENTIAL` | `MKL_THREADING_SEQUENTIAL` |
| Intel TBB threading | `TBB` | `MKL_THREADING_TBB` |

If the `mkl_set_threading_layer` function is called, the environment variable `MKL_THREADING_LAYER` is ignored.

**See Also**

Using the Single Dynamic Library
Layered Model Concept
Directory Structure in Detail

## Linking with Interface Libraries

### Using the ILP64 Interface vs. LP64 Interface

The Intel® oneAPI Math Kernel Library ILP64 libraries use the 64-bit integer type (necessary for indexing large arrays, with more than $2^{31}$-1 elements), whereas the LP64 libraries index arrays with the 32-bit integer type.

The LP64 and ILP64 interfaces are implemented in the Interface layer. Link with the following interface libraries for the LP64 or ILP64 interface, respectively:

- `libmkl_intel_lp64.a` or `libmkl_intel_ilp64.a` for static linking
- `libmkl_intel_lp64.dylib` or `libmkl_intel_ilp64.dylib` for dynamic linking

The ILP64 interface provides for the following:

- Support large data arrays (with more than $2^{31}$-1 elements)
- Enable compiling your Fortran code with the `-i8` compiler option

The LP64 interface provides compatibility with the previous Intel® oneAPI Math Kernel Library versions because "LP64" is just a new name for the only interface that the Intel® oneAPI Math Kernel Library versions lower than 9.1 provided. Choose the ILP64 interface if your application uses Intel® oneAPI Math Kernel Library for calculations with large data arrays or the library may be used so in the future.

On 64-bit platforms, selected domains provide API extensions with the `_64` suffix (for example, `SGEMM_64`) for supporting large data arrays in the LP64 library. This enables you to mix data types in one application. The selected domains and APIs include the following:

- BLAS: Fortran-style APIs for C applications and CBLAS APIs with integer arguments
- LAPACK: Fortran-style APIs for C applications and LAPACKE APIs with integer arguments

Intel® oneAPI Math Kernel Library provides the same include directory for the ILP64 and LP64 interfaces.

### Compiling for LP64/ILP64

The table below shows how to compile for the ILP64 and LP64 interfaces:

| Fortran | |
|---|---|
| Compiling for ILP64 | `ifort -i8 -I<mkl directory>/include ...` |
| Compiling for LP64 | `ifort -I<mkl directory>/include ...` |

| C or C++ | |
|---|---|
| Compiling for ILP64 | `icc -DMKL_ILP64 -I<mkl directory>/include ...` |
| Compiling for LP64 | `icc -I<mkl directory>/include ...` |

**Caution**
Linking of an application compiled with the `-i8` or `-DMKL_ILP64` option to the LP64 libraries may result in unpredictable consequences and erroneous output.

## Coding for ILP64

You do not need to change existing code if you are not using the ILP64 interface.

To migrate to ILP64 or write new code for ILP64, use appropriate types for parameters of the Intel® oneAPI Math Kernel Library functions and subroutines:

| Integer Types | Fortran | C or C++ |
|---|---|---|
| 32-bit integers | `INTEGER*4` or `INTEGER(KIND=4)` | `int` |
| Universal integers for ILP64/ LP64: <br>• 64-bit for ILP64 <br>• 32-bit otherwise | `INTEGER` without specifying `KIND` | `MKL_INT` |
| Universal integers for ILP64/ LP64: <br>• 64-bit integers | `INTEGER*8` or `INTEGER(KIND=8)` | `MKL_INT64` |
| FFT interface integers for ILP64/ LP64 | `INTEGER` without specifying `KIND` | `MKL_LONG` |

To determine the type of an integer parameter of a function, use appropriate include files. For functions that support only a Fortran interface, use the C/C++ include files `*.h`.

The above table explains which integer parameters of functions become 64-bit and which remain 32-bit for ILP64. The table applies to most Intel® oneAPI Math Kernel Library functions except some Vector Mathematics and Vector Statistics functions, which require integer parameters to be 64-bit or 32-bit regardless of the interface:

- **Vector Mathematics:** The *mode* parameter of the functions is 64-bit.
- **Random Number Generators (RNG):**

  All discrete RNG except `viRngUniformBits64` are 32-bit.

  The `viRngUniformBits64` generator function and `vslSkipAheadStream` service function are 64-bit.

- **Summary Statistics:** The *`estimate`* parameter of the `vslsSSCompute/vsldSSCompute` function is 64-bit.

Refer to the *Intel® oneAPI Math Kernel Library Developer Reference* for more information.

To better understand ILP64 interface details, see also examples.

### Limitations

All Intel® oneAPI Math Kernel Library function domains support ILP64 programming but FFTW interfaces to Intel® oneAPI Math Kernel Library:

- FFTW 2.x wrappers do not support ILP64.
- FFTW 3.x wrappers support ILP64 by a dedicated set of functions `plan_guru64`.

### See Also

High-level Directory Structure
Intel® oneAPI Math Kernel Library Include Files
Language Interfaces Support, by Function Domain
Layered Model Concept
Directory Structure in Detail

### Linking with Fortran 95 Interface Libraries

The `libmkl_blas95*.a` and `libmkl_lapack95*.a`libraries contain Fortran 95 interfaces for BLAS and LAPACK, respectively, which are compiler-dependent. In the Intel® oneAPI Math Kernel Library package, they are prebuilt for the Intel® Fortran compiler. If you are using a different compiler, build these libraries before using the interface.

### See Also

Fortran 95 Interfaces to LAPACK and BLAS
Compiler-dependent Functions and Fortran 90 Modules

## Linking with Threading Libraries

Intel® oneAPI Math Kernel Library threading layer defines how Intel® oneAPI Math Kernel Library functions utilize multiple computing cores of the system that the application runs on. You must link your application with one appropriate Intel® oneAPI Math Kernel Library library in this layer, as explained below. Depending on whether this is a threading or a sequential library, Intel® oneAPI Math Kernel Library runs in a parallel or sequential mode, respectively.

In the *parallel mode*, Intel® oneAPI Math Kernel Library utilizes multiple processor cores available on your system, uses the OpenMP*or Intel TBB threading technology, and requires a proper threading runtime library (RTL) to be linked with your application. Independently of use of Intel® oneAPI Math Kernel Library, the application may also require a threading RTL. You should link not more than one threading RTL to your application. Threading RTLs are provided by your compiler. Intel® oneAPI Math Kernel Library provides several threading libraries, each dependent on the threading RTL of a certain compiler, and your choice of the Intel® oneAPI Math Kernel Library threading library must be consistent with the threading RTL that you use in your application.

The OpenMP RTL of the Intel® compiler is the `libiomp5.dylib` library, located under *`<parent directory>`*`/compiler/lib`. This RTL is compatible with the GNU* compilers (gcc and gfortran). You can find additional information about the Intel OpenMP RTL at https://www.openmprtl.org.

The Intel TBB RTL of the Intel® compiler is the `libtbb.dylib` library, located under *`<parent directory>`*`/tbb/lib`. You can find additional information about the Intel TBB RTL at https://www.threadingbuildingblocks.org.

In the *sequential mode*, Intel® oneAPI Math Kernel Library runs unthreaded code, does not require an threading RTL, and does not respond to environment variables and functions controlling the number of threads. Avoid using the library in the sequential mode unless you have a particular reason for that, such as the following:

- Your application needs a threading RTL that none of Intel® oneAPI Math Kernel Library threading libraries is compatible with
- Your application is already threaded at a top level, and using parallel Intel® oneAPI Math Kernel Library only degrades the application performance by interfering with that threading
- Your application is intended to be run on a single thread, like a message-passing interface (MPI) application

It is critical to link the application with the proper RTL. The table below explains what library in the Intel® oneAPI Math Kernel Library threading layer and whatthreading RTL you should choose under different scenarios:

| **Application** | | **Intel® oneAPI Math Kernel Library** | | **RTL Required** |
|---|---|---|---|---|
| **Uses OpenMP** | **Compiled with** | **Execution Mode** | **Threading Layer** | |
| no | any compiler | parallel | Static linking: `libmkl_intel_ thread.a` Dynamic linking: `libmkl_intel_ thread.dylib` | `libiomp5.dylib` |
| no | any compiler | parallel | Static linking: `libmkl_tbb_ thread.a` Dynamic linking: `libmkl_tbb_ thread.dylib` | `libtbb.dylib` |
| no | any compiler | sequential | Static linking: `libmkl_ sequential.a` Dynamic linking: `libmkl_ sequential.dylib` | none[†] |
| yes | Intel compiler | parallel | Static linking: `libmkl_intel_ thread.a` Dynamic linking: `libmkl_intel_ thread.dylib` | `libiomp5.dylib` |
| yes | GNU compiler | parallel | Static linking: `libmkl_intel_ thread.a` Dynamic linking: `libmkl_intel_` | `libiomp5.dylib` |

| **Application** | | **Intel® oneAPI Math Kernel Library** | | **RTL Required** |
|---|---|---|---|---|
| **Uses OpenMP** | **Compiled with** | **Execution Mode** | **Threading Layer** | |
| | | | `thread.dylib` | |
| yes | any other compiler | parallel | Not supported. Use Intel® oneAPI Math Kernel Library in the sequential mode. | |

† For the sequential mode, add the POSIX threads library `(libpthread)` to your link line because the `libmkl_sequential.a` and `libmkl_sequential.dylib` libraries depend on `libpthread`.

**See Also**
Layered Model Concept
Notational Conventions

## Linking with Compiler Run-time Libraries

Dynamically link `libiomp5` or `libtbb` library even if you link other libraries statically.

Linking to the `libiomp5` statically can be problematic because the more complex your operating environment or application, the more likely redundant copies of the library are included. This may result in performance issues (oversubscription of threads) and even incorrect results.

To link `libiomp5` or `libtbb` dynamically, be sure the `DYLD_LIBRARY_PATH` environment variable is defined correctly.

**See Also**
Setting Environment Variables
Layered Model Concept

## Linking with System Libraries

To use the Intel® oneAPI Math Kernel Library FFT, Trigonometric Transform, or Poisson, Laplace, and HelmholtzSolver routines, link also the math support system library by adding "`-lm`" to the link line.

The `libiomp5` library relies on the native `pthread` library for multi-threading. Any time `libiomp5` is required, add `-lpthread`to your link line afterwards (the order of listing libraries is important).

The `libtbb` library relies on the compiler `libstdc++` library for C++ support. Any time `libtbb` is required, add `-lstdc++` to your link line afterwards (the order of listing libraries is important).

**NOTE**
To link with Intel® oneAPI Math Kernel Library statically using a GNU compiler, link also the system library`libdl` by adding `-ldl` to your link line. The Intel compiler always passes `-ldl` to the linker.

**See Also**
Linking Examples

## Building Custom Dynamically Linked Shared Libraries

Custom dynamically linked shared librariesreduce the collection of functions available in Intel® oneAPI Math Kernel Library libraries to those required to solve your particular problems, which helps to save disk space and build your own dynamic libraries for distribution.

The Intel® oneAPI Math Kernel Library customdynamically linked shared library builder enables you to create a dynamic ally linked shared library containing the selected functions and located in the `tools/builder` directory. The builder contains a makefile and a definition file with the list of functions.

### Using the Custom Dynamically Linked Shared Library Builder

To build a custom dynamically linked shared library, use the following command:

`make target [<options>]`

The following table lists possible values of `target` and explains what the command does for each value:

| Value | Comment |
|---|---|
| `libintel64` | The builder uses static Intel® oneAPI Math Kernel Library interface, threading, and core libraries to build an Intel 64 dynamically linked shared library. |
| `dylibuni` | The builder uses the single dynamic library `libmkl_rt.dylib` to build an Intel 64 dynamically linked shared library. |
| `help` | The command prints Help on the custom dynamically linked shared library builder |

The *<options>* placeholder stands for the list of parameters that define macros to be used by the makefile. The following table describes these parameters:

| Parameter [Values] | Description |
|---|---|
| `interface = {lp64\|ilp64}` | Defines whether to use LP64 or ILP64 programming interfacefor the Intel 64architecture.The default value is `lp64`. |
| `threading = {parallel\| sequential}` | Defines whether to use the Intel® oneAPI Math Kernel Library in the threaded or sequential mode. The default value is`parallel`. |
| `Prallel = {intel\|tbb}` | Specifies whether to use Intel OpenMP or Intel® oneTBB. The default value is`intel`. |
| `cluster = {yes\| no}` | (For `libintel64`only) Specifies whether Intel® oneAPI Math Kernel Library cluster components (BLACS, ScaLAPACK and/or CDFT) are needed to build the custom shared object. The default value is`no`. |
| `blacs_mpi = {intelmpi\| msmpi}` | Specifies the pre-compiled Intel® oneAPI Math Kernel Library BLACS library to use. Ignored if`'cluster=no'`. The default value is `intelmpi`. |
| `blacs_name = <lib name>` | Specifies the name (without extension) of a custom Intel® oneAPI Math Kernel Library BLACS library to use. Ignored if`'cluster=no'`. `'blacs_mpi'` is ignored if `'blacs_name'` was explicitly specified. The default value is `mkl_blacs_<blacs_mpi>_<interface>`. |
| `mpi = <lib name>` | Specifies the name (without extension) of the MPI library used to build the custom DLL. Ignored if `'cluster=no'`. The default value is `impi`. |

| Parameter [Values] | Description |
|---|---|
| `export = <file name>` | Specifies the full name of the file that contains the list of entry-point functions to be included in the shared object. The default name is `user_example_list` (no extension). |
| `name = <so name>` | Specifies the name of the library to be created. By default, the names of the created library is `mkl_custom.dylib`. |
| `xerbla = <error handler>` | Specifies the name of the object file `<user_xerbla>.o`that contains the error handler of the user. The makefile adds this error handler to the library for use instead of the default Intel® oneAPI Math Kernel Library error handler*`xerbla`*. If you omit this parameter, the native Intel® oneAPI Math Kernel Library*`xerbla`* is used. See the description of the *`xerbla`*function in the Intel® oneAPI Math Kernel Library Developer Reference to develop your own error handler. |
| `MKLROOT = <mkl directory>` | Specifies the location of Intel® oneAPI Math Kernel Library libraries used to build the customdynamically linked shared library. By default, the builder uses the Intel® oneAPI Math Kernel Library installation directory. |

All of the above parameters are optional. However, you must make the system and c-runtime (crt) libraries and link.exe available by setting the *PATH* and *LIB* environment variables appropriately. You can do this in the following ways:

- Manually
- If you are using the Intel compiler, use the `compilervars.sh` script with the appropriate 32-bit (x86) or 64-bit (x64 or amd-64) architecture flag.

In the simplest case, the command line is:

```
make ia32
```

and the missing options have default values. This command creates the `mkl_custom.dylib` library . The command takes the list of functions from the `user_list`file and uses the native Intel® oneAPI Math Kernel Library error handler*`xerbla`*.

Here is an example of a more complex case:

```
make intel64 export=my_func_list.txt name=mkl_small xerbla=my_xerbla.o
```

In this case, the command creates the `mkl_small.dylib` library. The command takes the list of functions from `my_func_list.txt` file and uses the error handler of the user `my_xerbla.o`.

**See Also**

Using the Single Dynamic Library

## Composing a List of Functions

To compose a list of functions for a minimal custom dynamically linked shared library needed for your application, you can use the following procedure:

1. Link your application with installed Intel® oneAPI Math Kernel Library libraries to make sure the application builds.
2. Remove all Intel® oneAPI Math Kernel Library libraries from the link line and start linking.

   Unresolved symbols indicate Intel® oneAPI Math Kernel Library functions that your application uses.
3. Include these functions in the list.

**Important**
Each time your application starts using more Intel® oneAPI Math Kernel Library functions, update the list to include the new functions.

## See Also

Specifying Function Names

## Specifying Function Names

In the file with the list of functions for your custom dynamically linked shared library, adjust function names to the required interface. For example, for Fortran functions append an underscore character "_" to the names as a suffix:

```
dgemm_
```

```
ddot_
```

```
dgetrf_
```

For more examples, see domain-specific lists of functions in the `<mkl directory>/tools/builder` folder.

**NOTE**
The lists of functions are provided in the `<mkl directory>/tools/builder` folder merely as examples. See Composing a List of Functions for how to compose lists of functions for your custom dynamically linked shared library.

**Tip**
Names of Fortran-style routines (BLAS, LAPACK, etc.) can be both upper-case or lower-case, with or without the trailing underscore. For example, these names are equivalent:
BLAS: `dgemm`, `DGEMM`, `dgemm_`, `DGEMM_`
LAPACK: `dgetrf`, `DGETRF`, `dgetrf_`, `DGETRF_`.

Properly capitalize names of C support functions in the function list. To do this, follow the guidelines below:

1. In the `mkl_service.h` include file, look up a `#define` directive for your function (`mkl_service.h` is included in the `mkl.h` header file).
2. Take the function name from the replacement part of that directive.

For example, the `#define` directive for the `mkl_disable_fast_mm` function is `#define mkl_disable_fast_mm MKL_Disable_Fast_MM`.

Capitalize the name of this function in the list like this: `MKL_Disable_Fast_MM`.

For the names of the Fortran support functions, see the tip.

**NOTE**
If selected functions have several processor-specific versions, the builder automatically includes them all in the custom library and the dispatcher manages them.

## Distributing Your Custom Dynamically Linked Shared Library

To enable use of your custom dynamically linked shared library in a threaded mode, distribute `libiomp5.dylib` along with the custom dynamically linked shared library.

# Managing Performance and Memory

4

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201

## Improving Performance with Threading

Intel® oneAPI Math Kernel Library (oneMKL) is extensively parallelized. SeeOpenMP* Threaded Functions and Problems and Functions Threaded with Intel® Threading Building Blocks for lists of threaded functions and problems that can be threaded.

Intel® oneAPI Math Kernel Library is*thread-safe*, which means that all Intel® oneAPI Math Kernel Library functions (except the LAPACK deprecated routine`?lacon)` work correctly during simultaneous execution by multiple threads. In particular, any chunk of threaded Intel® oneAPI Math Kernel Library code provides access for multiple threads to the same shared data, while permitting only one thread at any given time to access a shared piece of data. Therefore, you can call Intel® oneAPI Math Kernel Library from multiple threads and not worry about the function instances interfering with each other.

If you are using OpenMP* threading technology, you can use the environment variable `OMP_NUM_THREADS`to specify the number of threads or the equivalent OpenMP run-time function calls. Intel® oneAPI Math Kernel Library also offers variables that are independent of OpenMP, such as`MKL_NUM_THREADS`, and equivalent Intel® oneAPI Math Kernel Library functions for thread management. The Intel® oneAPI Math Kernel Library variables are always inspected first, then the OpenMP variables are examined, and if neither is used, the OpenMP software chooses the default number of threads.

By default, Intel® oneAPI Math Kernel Library uses the number ofOpenMP threads equal to the number of physical cores on the system.

If you are using the Intel TBB threading technology, the OpenMP threading controls, such as the `OMP_NUM_THREADS` environment variable or `MKL_NUM_THREADS` function, have no effect. Use the Intel TBB application programming interface to control the number of threads.

To achieve higher performance, set the number of threads to the number of processors or physical cores, as summarized in Techniques to Set the Number of Threads.

### OpenMP* Threaded Functions and Problems

The following Intel® oneAPI Math Kernel Library function domains are threaded with the OpenMP* technology:

- Direct sparse solver.
- LAPACK.

  For a list of threaded routines, see LAPACK Routines.
- Level1 and Level2 BLAS.

  For a list of threaded routines, see BLAS Level1 and Level2 Routines.
- All Level 3 BLAS and all Sparse BLAS routines except Level 2 Sparse Triangular solvers.
- All Vector Mathematics functions (except service functions).
- FFT.

  For a list of FFT transforms that can be threaded, see Threaded FFT Problems.

## LAPACK Routines

In this section, `?` stands for a precision prefix of *each* flavor of the respective routine and may have the value of `s, d, c`, or `z`.

The following LAPACK routines are threaded with OpenMP*:

- Linear equations, computational routines:
  - Factorization: `?getrf, ?getrfnpi, ?gbtrf, ?potrf, ?pptrf, ?sytrf, ?hetrf, ?sptrf, ?hptrf`
  - Solving: `?dttrsb, ?gbtrs, ?gttrs, ?pptrs, ?pbtrs, ?pttrs, ?sytrs, ?sptrs, ?hptrs, ?tptrs, ?tbtrs`
- Orthogonal factorization, computational routines:
  `?geqrf, ?ormqr, ?unmqr, ?ormlq, ?unmlq, ?ormql, ?unmql, ?ormrq, ?unmrq`
- Singular Value Decomposition, computational routines:
  `?gebrd, ?bdsqr`
- Symmetric Eigenvalue Problems, computational routines:
  `?sytrd, ?hetrd, ?sptrd, ?hptrd, ?steqr, ?stedc.`
- Generalized Nonsymmetric Eigenvalue Problems, computational routines:
  `chgeqz/zhgeqz.`

A number of other LAPACK routines, which are based on threaded LAPACK or BLAS routines, make effective use of OpenMP* parallelism:
`?gesv, ?posv, ?gels, ?gesvd, ?syev, ?heev, cgegs/zgegs, cgegv/zgegv, cgges/zgges, cggesx/zggesx, cggev/zggev, cggevx/zggevx`, and so on.

## Threaded BLAS Level1 and Level2 Routines

In the following list, `?` stands for a precision prefix of *each* flavor of the respective routine and may have the value of `s, d, c`, or `z`.

The following routines are threaded with OpenMP*:

- Level1 BLAS:
  `?axpy, ?copy, ?swap, ddot/sdot, cdotc, drot/srot`
- Level2 BLAS:
  `?gemv, ?trsv, ?trmv, dsyr/ssyr, dsyr2/ssyr2, dsymv/ssymv`

## Threaded FFT Problems

The following characteristics of a specific problem determine whether your FFT computation may be threaded with OpenMP*:

- rank
- domain
- size/length
- precision (single or double)
- placement (in-place or out-of-place)
- strides
- number of transforms
- layout (for example, interleaved or split layout of complex data)

Most FFT problems are threaded. In particular, computation of multiple transforms in one call (number of transforms > 1) is threaded. Details of which transforms are threaded follow.

**One-dimensional (1D) transforms**

1D transforms are threaded in many cases.

1D complex-to-complex (c2c) transforms of size *N* using interleaved complex data layout are threaded under the following conditions depending on the architecture:

| Architecture | Conditions |
|---|---|
| Intel® 64 | *N* is a power of 2, $log_2(N) > 9$, the transform is double-precision out-of-place, and input/output strides equal 1. |
| Any | *N* is composite, $log_2(N) > 16$, and input/output strides equal 1. |

1D complex-to-complex transforms using split-complex layout are not threaded.

**Multidimensional transforms**

All multidimensional transforms on large-volume data are threaded.

## Functions Threaded with Intel® Threading Building Blocks

In this section, `?` stands for a precision prefix or suffix of the routine name and may have the value of `s`, `d`, `c`, or `z`.

The following Intel® oneAPI Math Kernel Library function domains are threaded with Intel® Threading Building Blocks (oneTBB):

- LAPACK.

  For a list of threaded routines, see LAPACK Routines.
- Entire Level3 BLAS.
- Level2 BLAS – ?GEMV.
- Fast Poisson, Laplace, and Helmholtz Solver (Poisson Library).
- All Vector Mathematics functions (except service functions).
- Intel® oneAPI Math Kernel Library PARDISO, a direct sparse solver based on Parallel Direct Sparse Solver (PARDISO*).

  For details, see oneMKL PARDISO Steps.
- Sparse BLAS.

  For a list of threaded routines, see Sparse BLAS Routines.

### LAPACK Routines

The following LAPACK routines are threaded with oneTBB:
`?geqrf`, `?gelqf`, `?getrf`, `?potrf`, `?unmqr*`, `?ormqr*`, `?unmrq*`, `?ormrq*`, `?unmlq*`, `?ormlq*`, `?unmql*`, `?ormql*`, `?sytrd`, `?hetrd`, `?syev`, `?heev`, and `?latrd`.

A number of other LAPACK routines, which are based on threaded LAPACK or BLAS routines, make effective use of oneTBB threading:
`?getrs`, `?gesv`, `?potrs`, `?bdsqr`, and `?gels`.

### oneMKL PARDISO Steps

Intel® oneAPI Math Kernel Library PARDISO is threaded with oneTBB in the reordering and factorization steps. However, routines performing the solving step are still called sequentially when using oneTBB.

### Sparse BLAS Routines

The Sparse BLAS inspector-executor application programming interface routines `mkl_sparse_?_mv` are threaded with oneTBB for the general compressed sparse row (CSR) and block sparse row (BSR) formats.

The following Sparse BLAS inspector-executor application programming routines are threaded with oneTBB:

- `mkl_sparse_?_mv` using the general compressed sparse row (CSR) and block sparse row (BSR) matrix formats.

- `mkl_sparse_?_mm` using the general CSR sparse matrix format and both row and column major storage formats for the dense matrix.

## Avoiding Conflicts in the Execution Environment

Certain situations can cause conflicts in the execution environment that make the use of threads in Intel® oneAPI Math Kernel Library problematic. This section briefly discusses why these problems exist and how to avoid them.

If your program is parallelized by means other than Intel® OpenMP* run-time library (RTL) and Intel® Threading Building Blocks (oneTBB) RTL, several calls to Intel® oneAPI Math Kernel Library may operate in a multithreaded mode at the same time and result in slow performance due to overuse of machine resources.

The following table considers several cases where the conflicts may arise and provides recommendations depending on your threading model:

| Threading model | Discussion |
|---|---|
| You parallelize the program using the technology other than Intel OpenMP and oneTBB (for example: `pthreads` on macOS*). | If more than one thread calls Intel® oneAPI Math Kernel Library, and the function being called is threaded, it may be important that you turn off Intel® oneAPI Math Kernel Library threading. Set the number of threads to one by any of the available means (seeTechniques to Set the Number of Threads). |
| You parallelize the program using OpenMP directives and/or pragmas and compile the program using a non-Intel compiler. | To avoid simultaneous activities of multiple threading RTLs, link the program against the Intel® oneAPI Math Kernel Library threading library that matches the compiler you use (see Linking Exampleson how to do this). If this is not possible, use Intel® oneAPI Math Kernel Library in the sequential mode. To do this, you should link with the appropriate threading library:`libmkl_sequential.a` or `libmkl_sequential.dylib` (see Appendix C: Directory Structure in Detail). |
| You thread the program using oneTBB threading technology and compile the program using a non-Intel compiler. | To avoid simultaneous activities of multiple threading RTLs, link the program against the Intel® oneAPI Math Kernel Library oneTBB threading library and oneTBB RTL if it matches the compiler you use. If this is not possible, use Intel® oneAPI Math Kernel Library in the sequential mode. To do this, link with the appropriate threading library:`libmkl_sequential.a` or `libmkl_sequential.dylib` (see Appendix C: Directory Structure in Detail). |
| You run multiple programs calling Intel® oneAPI Math Kernel Library on a multiprocessor system, for example, a program parallelized using a message-passing interface (MPI). | The threading RTLs from different programs you run may place a large number of threads on the same processor on the system and therefore overuse the machine resources. In this case, one of the solutions is to set the number of threads to one by any of the available means (see Techniques to Set the Number of Threads). |

Using the `mkl_set_num_threads` and `mkl_domain_set_num_threads`functions to control parallelism of Intel® oneAPI Math Kernel Library from parallel user threads may result in a race condition that impacts the performance of the application because these functions operate on internal control variables that are global, that is, apply to all threads. For example, if parallel user threads call these functions to set different numbers of threads for the same function domain, the number of threads actually set is unpredictable. To avoid this kind of data races, use the`mkl_set_num_threads_local` function (see the "Support Functions" chapter in the *Intel® oneAPI Math Kernel Library Developer Reference* for the function description).

### See Also

Using Additional Threading Control

Linking with Compiler Support RTLs

## Techniques to Set the Number of Threads

Use the following techniques to specify the number of OpenMP threads to use in Intel® oneAPI Math Kernel Library:

- Set one of the OpenMP or Intel® oneAPI Math Kernel Library environment variables:
  - `OMP_NUM_THREADS`
  - `MKL_NUM_THREADS`
  - `MKL_DOMAIN_NUM_THREADS`
- Call one of the OpenMP or Intel® oneAPI Math Kernel Library functions:
  - `omp_set_num_threads()`
  - `mkl_set_num_threads()`
  - `mkl_domain_set_num_threads()`
  - `mkl_set_num_threads_local()`

**NOTE**
A call to the `mkl_set_num_threads` or `mkl_domain_set_num_threads`function changes the number of OpenMP threads available to all in-progress calls (in concurrent threads) and future calls to Intel® oneAPI Math Kernel Library and may result in slow Intel® oneAPI Math Kernel Library performance and/or race conditions reported by run-time tools, such as Intel® Inspector.

To avoid such situations, use the mkl_set_num_threads_local function (see the "Support Functions" section in the *Intel® oneAPI Math Kernel Library Developer Reference* for the function description).

When choosing the appropriate technique, take into account the following rules:

- The Intel® oneAPI Math Kernel Library threading controls take precedence over the OpenMP controls because they are inspected first.
- A function call takes precedence over any environment settings. The exception, which is a consequence of the previous rule, is that a call to the OpenMP subroutine `omp_set_num_threads()` does not have precedence over the settings of Intel® oneAPI Math Kernel Library environment variables such as`MKL_NUM_THREADS`. See Using Additional Threading Control for more details.
- You cannot change run-time behavior in the course of the run using the environment variables because they are read only once at the first call to Intel® oneAPI Math Kernel Library.

If you use the Intel TBB threading technology, read the documentation for the `tbb::task_scheduler_init` class at https://www.threadingbuildingblocks.org/documentation to find out how to specify the number of threads.

## Setting the Number of Threads Using an OpenMP* Environment Variable

You can set the number of threads using the environment variable `OMP_NUM_THREADS`. To change the number of OpenMP threads, in the command shell in which the program is going to run, enter:

`export OMP_NUM_THREADS=<`*`number of threads to use`*`>`.

### See Also

Using Additional Threading Control

## Changing the Number of OpenMP* Threads at Run Time

You cannot change the number of OpenMP threads at run time using environment variables. However, you can call OpenMP routines to do this. Specifically, the following sample code shows how to change the number of threads during run time using the `omp_set_num_threads()` routine. For more options, see also Techniques to Set the Number of Threads.

The example is provided for both C and Fortran languages. To run the example in C, use the `omp.h` header file from the Intel(R) compiler package. If you do not have the Intel compiler but wish to explore the functionality in the example, use Fortran API for `omp_set_num_threads()` rather than the C version. For example, `omp_set_num_threads_( &i_one );`

```
// ******* C language *******
#include "omp.h"
#include "mkl.h"
#include <stdio.h>
#define SIZE 1000
int main(int args, char *argv[]){
double *a, *b, *c;
a = (double*)malloc(sizeof(double)*SIZE*SIZE);
b = (double*)malloc(sizeof(double)*SIZE*SIZE);
c = (double*)malloc(sizeof(double)*SIZE*SIZE);
double alpha=1, beta=1;
int m=SIZE, n=SIZE, k=SIZE, lda=SIZE, ldb=SIZE, ldc=SIZE, i=0, j=0;
char transa='n', transb='n';
for( i=0; i<SIZE; i++)
{
    for( j=0; j<SIZE; j++)
    {
        a[i*SIZE+j]= (double)(i+j);
        b[i*SIZE+j]= (double)(i*j);
        c[i*SIZE+j]= (double)0;
    }
}
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);
printf("row\ta\tc\n");
for ( i=0;i<10;i++)
{
printf("%d:\t%f\t%f\n", i, a[i*SIZE], c[i*SIZE]);
}
omp_set_num_threads(1);
for( i=0; i<SIZE; i++)
{
    for( j=0; j<SIZE; j++)
    {
        a[i*SIZE+j]= (double)(i+j);
        b[i*SIZE+j]= (double)(i*j);
        c[i*SIZE+j]= (double)0;
    }
}
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);
printf("row\ta\tc\n");
for ( i=0;i<10;i++)
{
printf("%d:\t%f\t%f\n", i, a[i*SIZE], c[i*SIZE]);
}
omp_set_num_threads(2);
for( i=0; i<SIZE; i++)
{
    for( j=0; j<SIZE; j++)
    {
        a[i*SIZE+j]= (double)(i+j);
        b[i*SIZE+j]= (double)(i*j);
        c[i*SIZE+j]= (double)0;
    }
}
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);
printf("row\ta\tc\n");
for ( i=0;i<10;i++)
{
printf("%d:\t%f\t%f\n", i, a[i*SIZE],
c[i*SIZE]);
}
free (a);
free (b);
free (c);
return 0;
}
```

```
// ******* Fortran language *******
PROGRAM DGEMM_DIFF_THREADS
INTEGER N, I, J
PARAMETER (N=100)
REAL*8 A(N,N),B(N,N),C(N,N)
REAL*8 ALPHA, BETA

ALPHA = 1.1
BETA = -1.2
DO I=1,N
    DO J=1,N
        A(I,J) = I+J
        B(I,J) = I*j
        C(I,J) = 0.0
    END DO
END DO
CALL DGEMM('N','N',N,N,N,ALPHA,A,N,B,N,BETA,C,N)
print *,'Row A C'
DO i=1,10
write(*,'(I4,F20.8,F20.8)') I, A(1,I),C(1,I)
END DO
CALL OMP_SET_NUM_THREADS(1);
DO I=1,N
    DO J=1,N
        A(I,J) = I+J
        B(I,J) = I*j
        C(I,J) = 0.0
    END DO
END DO
CALL DGEMM('N','N',N,N,N,ALPHA,A,N,B,N,BETA,C,N)
print *,'Row A C'
DO i=1,10
write(*,'(I4,F20.8,F20.8)') I, A(1,I),C(1,I)
END DO
CALL OMP_SET_NUM_THREADS(2);
DO I=1,N
    DO J=1,N
        A(I,J) = I+J
        B(I,J) = I*j
        C(I,J) = 0.0
    END DO
END DO
CALL DGEMM('N','N',N,N,N,ALPHA,A,N,B,N,BETA,C,N)
print *,'Row A C'
DO i=1,10
write(*,'(I4,F20.8,F20.8)') I, A(1,I),C(1,I)
END DO
STOP
END
```

## Using Additional Threading Control

### oneMKL-specific Environment Variables for OpenMP Threading Control

Intel® oneAPI Math Kernel Library provides environment variables and support functions to control Intel® oneAPI Math Kernel Library threading independently of OpenMP. The Intel® oneAPI Math Kernel Library-specific threading controls take precedence over their OpenMP equivalents. Use the Intel® oneAPI Math Kernel Library-specific threading controls to distribute OpenMP threads between Intel® oneAPI Math Kernel Library and the rest of your program.

**NOTE**
Some Intel® oneAPI Math Kernel Library routines may use fewer OpenMP threads than suggested by the threading controls if either the underlying algorithms do not support the suggested number of OpenMP threads or the routines perform better with fewer OpenMP threads because of lower OpenMP overhead and/or better data locality. Set the`MKL_DYNAMIC` environment variable to `FALSE` or call `mkl_set_dynamic(0)` to use the suggested number of OpenMP threads whenever the algorithms permit and regardless of OpenMP overhead and data locality.

The table below lists the Intel® oneAPI Math Kernel Library environment variables for threading control, their equivalent functions, and OMP counterparts:

| Environment Variable | Support Function | Comment | Equivalent OpenMP* Environment Variable |
|---|---|---|---|
| `MKL_NUM_THREADS` | `mkl_set_num_threads` `mkl_set_num_threads_local` | Suggests the number of OpenMP threads to use. | `OMP_NUM_THREADS` |
| `MKL_DOMAIN_NUM_THREADS` | `mkl_domain_set_num_threads` | Suggests the number of OpenMP threads for a particular function domain. | |
| `MKL_DYNAMIC` | `mkl_set_dynamic` | Enables Intel® oneAPI Math Kernel Library to dynamically change the number of OpenMP threads. | `OMP_DYNAMIC` |

**NOTE**
Call mkl_set_num_threads() to force Intel® oneAPI Math Kernel Library to use a given number of OpenMP threads and prevent it from reacting to the environment variablesMKL_NUM_THREADS, MKL_DOMAIN_NUM_THREADS, and OMP_NUM_THREADS.

The example below shows how to force Intel® oneAPI Math Kernel Library to use one thread:

```
// ******* C language *******

#include <mkl.h>
...
mkl_set_num_threads ( 1 );
```

```
// ******* Fortran language *******
...
call mkl_set_num_threads( 1 )
```

## MKL_DYNAMIC

The MKL_DYNAMICenvironment variable enables Intel® oneAPI Math Kernel Library to dynamically change the number of threads.

The default value of MKL_DYNAMIC is TRUE, regardless of OMP_DYNAMIC, whose default value may be FALSE.

When MKL_DYNAMIC is TRUE, Intel® oneAPI Math Kernel Library may use fewer OpenMP threads than the maximum number you specify.

For example, MKL_DYNAMIC set to TRUE enables optimal choice of the number of threads in the following cases:

- If the requested number of threads exceeds the number of physical cores (perhaps because of using the Intel® Hyper-Threading Technology), Intel® oneAPI Math Kernel Library scales down the number of OpenMP threads to the number of physical cores.
- If you are able to detect the presence of a message-passing interface (MPI), but cannot determine whether it has been called in a thread-safe mode, Intel® oneAPI Math Kernel Library runs one OpenMP thread.

When MKL_DYNAMIC is FALSE, Intel® oneAPI Math Kernel Library uses the suggested number of OpenMP threads whenever the underlying algorithms permit.For example, if you attempt to do a size one matrix-matrix multiply across eight threads, the library may instead choose to use only one thread because it is impractical to use eight threads in this event.

If Intel® oneAPI Math Kernel Library is called from an OpenMP parallel region in your program, Intel® oneAPI Math Kernel Library uses only one thread by default. If you want Intel® oneAPI Math Kernel Library to go parallel in such a call, link your program against an OpenMP threading RTL supported by Intel® oneAPI Math Kernel Library and set the environment variables:

- OMP_NESTED to TRUE
- OMP_DYNAMIC and MKL_DYNAMIC to FALSE
- MKL_NUM_THREADS to some reasonable value

With these settings, Intel® oneAPI Math Kernel Library usesMKL_NUM_THREADS threads when it is called from the OpenMP parallel region in your program.

In general, set MKL_DYNAMIC to FALSEonly under circumstances that Intel® oneAPI Math Kernel Library is unable to detect, for example, to use nested parallelism where the library is already called from a parallel section.

## MKL_DOMAIN_NUM_THREADS

The `MKL_DOMAIN_NUM_THREADS` environment variable suggests the number of OpenMP threads for a particular function domain.

`MKL_DOMAIN_NUM_THREADS` accepts a string value `<MKL-env-string>`, which must have the following format:

*<MKL-env-string> ::= <MKL-domain-env-string> { <delimiter><MKL-domain-env-string> }*

*<delimiter> ::= [ <space-symbol>* ] ( <space-symbol> | <comma-symbol> | <semicolon-symbol> | <colon-symbol>) [ <space-symbol>* ]*

*<MKL-domain-env-string> ::= <MKL-domain-env-name><uses><number-of-threads>*

*<MKL-domain-env-name>* ::= `MKL_DOMAIN_ALL | MKL_DOMAIN_BLAS | MKL_DOMAIN_FFT | MKL_DOMAIN_VML | MKL_DOMAIN_PARDISO`

*<uses> ::= [ <space-symbol>* ] ( <space-symbol> | <equality-sign> | <comma-symbol>) [ <space-symbol>* ]*

*<number-of-threads> ::= <positive-number>*

*<positive-number> ::= <decimal-positive-number> | <octal-number> | <hexadecimal-number>*

In the syntax above, values of *<MKL-domain-env-name>* indicate function domains as follows:

| | |
|---|---|
| `MKL_DOMAIN_ALL` | All function domains |
| `MKL_DOMAIN_BLAS` | BLAS Routines |
| `MKL_DOMAIN_FFT` | Fourier Transform Functions |
| `MKL_DOMAIN_LAPACK` | LAPACK Routines |
| `MKL_DOMAIN_VML` | Vector Mathematics (VM) |
| `MKL_DOMAIN_PARDISO` | Intel® oneAPI Math Kernel Library PARDISO, a direct sparse solver based on Parallel Direct Sparse Solver (PARDISO*) |

For example, you could set the `MKL_DOMAIN_NUM_THREADS`environment variable to any of the following string variants, in this case, defining three specific domain variables internal to Intel® oneAPI Math Kernel Library:

```
MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL=2, MKL_DOMAIN_BLAS=1, MKL_DOMAIN_FFT=4"
MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL 2 : MKL_DOMAIN_BLAS 1 : MKL_DOMAIN_FFT 4"
MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL=2 : MKL_DOMAIN_BLAS=1 : MKL_DOMAIN_FFT=4"
MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL=2; MKL_DOMAIN_BLAS=1; MKL_DOMAIN_FFT=4"
MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL=2 MKL_DOMAIN_BLAS 1, MKL_DOMAIN_FFT 4"
MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL,2: MKL_DOMAIN_BLAS 1, MKL_DOMAIN_FFT,4"
```

**NOTE** Prepend the appropriate `set`/`export`/`setenv` command for your command shell and operating system. Refer to Setting the Environment Variables for Threading Control for more details.

The global variables `MKL_DOMAIN_ALL`, `MKL_DOMAIN_BLAS`, `MKL_DOMAIN_FFT`, `MKL_DOMAIN_VML`, and `MKL_DOMAIN_PARDISO`, as well as the interface for the Intel® oneAPI Math Kernel Library threading control functions, can be found in the`mkl.h` header file.

**NOTE** You can retrieve the values of the specific domain variables that you have set in your code with a call to the `mkl_get_domain_max_threads(domain_name)` function per the Fortran and C interface with the desired domain variable name.

This table illustrates how values of `MKL_DOMAIN_NUM_THREADS` are interpreted.

| **Value of `MKL_DOMAIN_NUM_THREADS`** | **Interpretation** |
|---|---|
| `MKL_DOMAIN_ALL=4` | All parts of Intel® oneAPI Math Kernel Library should try four OpenMP threads. The actual number of threads may be still different because of the`MKL_DYNAMIC` setting or system resource issues. The setting is equivalent to `MKL_NUM_THREADS` = 4. |
| `MKL_DOMAIN_ALL=1, MKL_DOMAIN_BLAS=4` | All parts of Intel® oneAPI Math Kernel Library should try one OpenMP thread, except for BLAS, which is suggested to try four threads. |
| `MKL_DOMAIN_VML=2` | VM should try two OpenMP threads. The setting affects no other part of Intel® oneAPI Math Kernel Library. |

Be aware that the domain-specific settings take precedence over the overall ones. For example, the "`MKL_DOMAIN_BLAS=4`" value of `MKL_DOMAIN_NUM_THREADS` suggests trying four OpenMP threads for BLAS, regardless of later setting `MKL_NUM_THREADS`, and a function call "`mkl_domain_set_num_threads ( 4, MKL_DOMAIN_BLAS );`" suggests the same, regardless of later calls to `mkl_set_num_threads()`. However, a function call with input "`MKL_DOMAIN_ALL`", such as "`mkl_domain_set_num_threads (4, MKL_DOMAIN_ALL);`" is equivalent to "`mkl_set_num_threads(4)`", and thus it will be overwritten by later calls to `mkl_set_num_threads`. Similarly, the environment setting of `MKL_DOMAIN_NUM_THREADS` with "`MKL_DOMAIN_ALL=4`" will be overwritten with `MKL_NUM_THREADS` = 2.

Whereas the `MKL_DOMAIN_NUM_THREADS` environment variable enables you set several variables at once, for example, "`MKL_DOMAIN_BLAS=4,MKL_DOMAIN_FFT=2`", the corresponding function does not take string syntax. So, to do the same with the function calls, you may need to make several calls, which in this example are as follows:

```
mkl_domain_set_num_threads ( 4, MKL_DOMAIN_BLAS );

mkl_domain_set_num_threads ( 2, MKL_DOMAIN_FFT );
```

## MKL_NUM_STRIPES

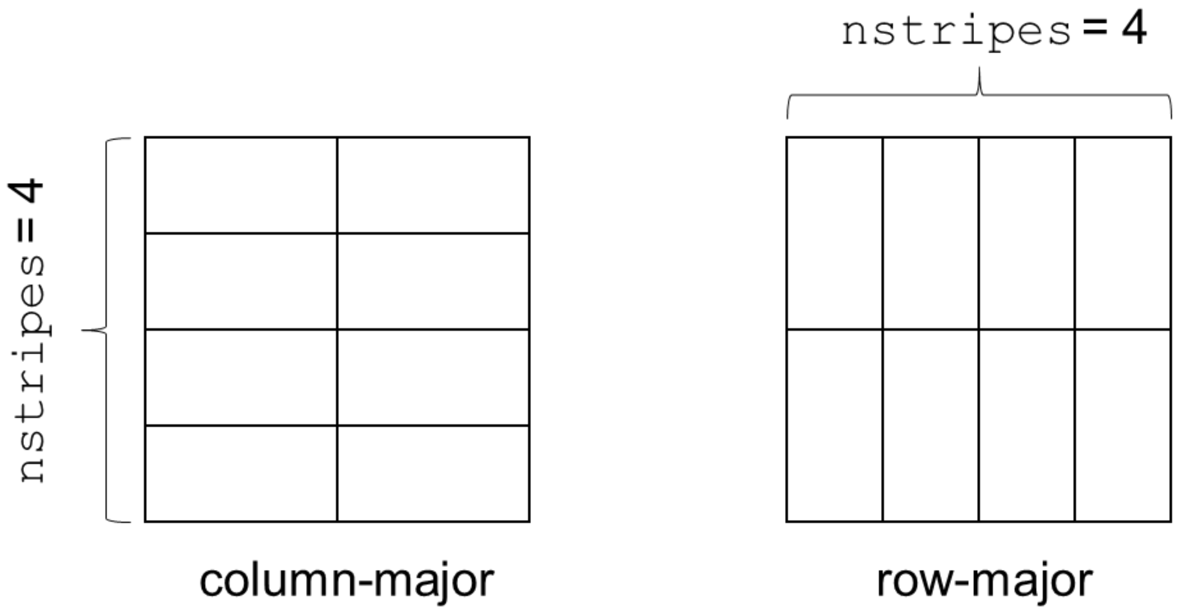
The `MKL_NUM_STRIPES`environment variable controls the Intel® oneAPI Math Kernel Library threading algorithm for`?gemm` functions. When `MKL_NUM_STRIPES` is set to a positive integer value *nstripes*, Intel® oneAPI Math Kernel Library tries to use a number of partitions equal to*nstripes* along the leading dimension of the output matrix.

The following table explains how the value *nstripes* of `MKL_NUM_STRIPES`defines the partitioning algorithm used by Intel® oneAPI Math Kernel Library for`?gemm` output matrix; *max_threads_for_mkl*denotes the maximum number of OpenMP threads for Intel® oneAPI Math Kernel Library:

| **Value of `MKL_NUM_STRIPES`** | **Partitioning Algorithm** |
|---|---|
| 1 < *nstripes* < (*max_threads_for_mkl*/ 2) | 2D partitioning with the number of partitions equal to *nstripes*: <br>• Horizontal, for column-major ordering. <br>• Vertical, for row-major ordering. |
| *nstripes* = 1 | 1D partitioning algorithm along the opposite direction of the leading dimension. |

| Value of `MKL_NUM_STRIPES` | Partitioning Algorithm |
|---|---|
| *nstripes* ≥ (*max_threads_for_mkl* / 2) | 1D partitioning algorithm along the leading dimension. |
| *nstripes* < 0 | The default Intel® oneAPI Math Kernel Library threading algorithm. |

The following figure shows the partitioning of an output matrix for *nstripes* = 4 and a total number of 8 OpenMP threads for column-major and row-major orderings:



You can use support functions `mkl_set_num_stripes` and `mkl_get_num_stripes` to set and query the number of stripes, respectively.

## Setting the Environment Variables for Threading Control

To set the environment variables used for threading control, in the command shell in which the program is going to run, enter:

```
export <VARIABLE NAME>=<value>
```

For example:

```
export MKL_NUM_THREADS=4

export MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL=1, MKL_DOMAIN_BLAS=4"

export MKL_DYNAMIC=FALSE

export MKL_NUM_STRIPES=4
```

## Calling oneMKL Functions from Multi-threaded Applications

This section summarizes typical usage models and available options for calling Intel® oneAPI Math Kernel Library functions from multi-threaded applications. These recommendations apply to any multi-threading environments: OpenMP*, Intel® Threading Building Blocks,POSIX* threads, and others.

## Usage model: disable oneMKL internal threading for the whole application

**When used:**Intel® oneAPI Math Kernel Library internal threading interferes with application's own threading or may slow down the application.

**Example:** the application is threaded at top level, or the application runs concurrently with other applications.

**Options:**

- Link statically or dynamically with the sequential library
- Link with the Single Dynamic Library `mkl_rt.dylib` and select the sequential library using an environment variable or a function call:
  - Set `MKL_THREADING_LAYER=sequential`
  - Call `mkl_set_threading_layer(MKL_THREADING_SEQUENTIAL)` ‡

## Usage model: partition system resources among application threads

**When used:** application threads are specialized for a particular computation.

**Example:** one thread solves equations on all cores but one, while another thread running on a single core updates a database.

**Linking Options:**

- Link statically or dynamically with a threading library
- Link with the Single Dynamic Library `mkl_rt.dylib` and select a threading library using an environment variable or a function call:
  - set `MKL_THREADING_LAYER=intel` or `MKL_THREADING_LAYER=tbb`
  - call `mkl_set_threading_layer(MKL_THREADING_INTEL)` or `mkl_set_threading_layer(MKL_THREADING_TBB)`

**Other Options for OpenMP Threading:**

- Set the `MKL_NUM_THREADS`environment variable to a desired number of OpenMP threads for Intel® oneAPI Math Kernel Library.
- Set the `MKL_DOMAIN_NUM_THREADS`environment variable to a desired number of OpenMP threads for Intel® oneAPI Math Kernel Library for a particular function domain.

  Use if the application threads work with different Intel® oneAPI Math Kernel Library function domains.
- Call `mkl_set_num_threads()`

  Use to globally set a desired number of OpenMP threads for Intel® oneAPI Math Kernel Library at run time.
- Call `mkl_domain_set_num_threads()`.

  Use if at some point application threads start working with different Intel® oneAPI Math Kernel Library function domains.
- Call `mkl_set_num_threads_local()`.

  Use to set the number of OpenMP threads for Intel® oneAPI Math Kernel Library called from a particular thread.

**NOTE**
If your application uses OpenMP* threading, you may need to provide additional settings:

- Set the environment variable `OMP_NESTED=TRUE`, or alternatively call `omp_set_nested(1)`, to enable OpenMP nested parallelism.
- Set the environment variable `MKL_DYNAMIC=FALSE`, or alternatively call `mkl_set_dynamic(0)`, to prevent Intel® oneAPI Math Kernel Library from dynamically reducing the number of OpenMP threads in nested parallel regions.

‡ For details of the mentioned functions, see the Support Functions section of the *Intel® oneAPI Math Kernel Library Developer Reference*, available in the Intel Software Documentation Library.

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.<br><br>Notice revision #20201201 |

**See Also**
Linking with Threading Libraries
Dynamically Selecting the Interface and Threading Layer
oneMKL-specific Environment Variables for OpenMP Threading Control
`MKL_DOMAIN_NUM_THREADS`
Avoiding Conflicts in the Execution Environment
Intel Software Documentation Library

## Using Intel® Hyper-Threading Technology

Intel® Hyper-Threading Technology (Intel® HT Technology) is especially effective when each thread performs different types of operations and when there are under-utilized resources on the processor. However, Intel® oneAPI Math Kernel Library fits neither of these criteria because the threaded portions of the library execute at high efficiencies using most of the available resources and perform identical operations on each thread. You may obtain higher performance by disabling Intel HT Technology.

If you run with Intel HT Technology enabled, performance may be especially impacted if you run on fewer threads than physical cores. Moreover, if, for example, there are two threads to every physical core, the thread scheduler may assign two threads to some cores and ignore the other cores altogether. If you are using the OpenMP* library of the Intel Compiler, read the respective User Guide on how to best set the thread affinity interface to avoid this situation. For Intel® oneAPI Math Kernel Library, apply the following setting:

```
export KMP_AFFINITY=granularity=fine,compact,1,0
```

If you are using the Intel TBB threading technology, read the documentation on the `tbb::affinity_partitioner` class at https://www.threadingbuildingblocks.org/documentation to find out how to affinitize Intel TBB threads.

## Improving Performance for Small Size Problems

The overhead of calling an Intel® oneAPI Math Kernel Library function for small problem sizes can be significant when the functionhas a large number of parameters or internally checks parameter errors. To reduce the performance overhead for these small size problems, the Intel® oneAPI Math Kernel Library*direct call*feature works in conjunction with the compiler to preprocess the calling parameters to supported Intel® oneAPI Math Kernel Library functions and directly call or inline special optimized small-matrix kernels that bypass error checking.For a list of functions supporting direct call, see Limitations of the Direct Call.

To activate the feature, do the following:

- Compile your C or Fortran code with the preprocessor macro depending on whether a threaded or sequential mode of Intel® oneAPI Math Kernel Library is required by supplying the compiler option as explained below:

| **Intel® oneAPI Math Kernel Library Mode** | **Macro** | **Compiler Option** |
|---|---|---|
| Threaded | `MKL_DIRECT_CALL` | `-DMKL_DIRECT_CALL` |
| Sequential | `MKL_DIRECT_CALL_SEQ` | `-DMKL_DIRECT_CALL_SEQ` |

- For Fortran applications:
  - Enable preprocessor by using the `-fpp` option for Intel® Fortran Compiler.
  - Include the Intel® oneAPI Math Kernel Library Fortran include file`mkl_direct_call.fi`.

Intel® oneAPI Math Kernel Library skips error checking and intermediate function calls if the problem size is small enough (for example: a call to a function that supports direct call, such as`dgemm`, with matrix ranks smaller than 50).

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. |
| Notice revision #20201201 |

## Using MKL_DIRECT_CALL in C Applications

The following examples of code and link lines show how to activate direct calls to Intel® oneAPI Math Kernel Library kernels in C applications:

- Include the `mkl.h` header file:

```
#include "mkl.h"
int main(void) {

// Call Intel MKL DGEMM

return 0;
}
```

- For multi-threaded Intel® oneAPI Math Kernel Library, compile with`MKL_DIRECT_CALL` preprocessor macro:

```
icc -DMKL_DIRECT_CALL -std=c99 your_application.c $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a
$(MKLROOT)/lib/intel64/libmkl_intel_thread.a -lpthread -lm -openmp -I$(MKLROOT)/include
```

- To use Intel® oneAPI Math Kernel Library in the sequential mode, compile with`MKL_DIRECT_CALL_SEQ` preprocessor macro:

```
icc -DMKL_DIRECT_CALL_SEQ -std=c99 your_application.c $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a
$(MKLROOT)/lib/intel64/libmkl_sequential.a -lpthread -lm -I$(MKLROOT)/include
```

## Using MKL_DIRECT_CALL in Fortran Applications

The following examples of code and link lines show how to activate direct calls to Intel® oneAPI Math Kernel Library kernels in Fortran applications:

- Include `mkl_direct_call.fi`, to be preprocessed by the Fortran compiler preprocessor

```
#     include "mkl_direct_call.fi"
      program   DGEMM_MAIN
....
*      Call Intel MKL DGEMM
....
      call sub1()
      stop 1
      end

*     A subroutine that calls DGEMM
      subroutine sub1
*      Call Intel MKL DGEMM

      end
```

- For multi-threaded Intel® oneAPI Math Kernel Library, compile with-`fpp` option for Intel Fortran compiler and with `MKL_DIRECT_CALL` preprocessor macro:

```
ifort -DMKL_DIRECT_CALL -fpp your_application.f $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a $(MKLROOT)/lib/intel64/libmkl_intel_thread.a
-lpthread -lm -openmp -I$(MKLROOT)/include
```

- To use Intel® oneAPI Math Kernel Library in the sequential mode, compile with-`fpp` option for Intel Fortran compiler (or with `-Mpreprocess` for PGI compilers) and with `MKL_DIRECT_CALL_SEQ` preprocessor macro:

```
ifort -DMKL_DIRECT_CALL_SEQ -fpp your_application.f $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a $(MKLROOT)/lib/intel64/libmkl_sequential.a
-lpthread -lm -I$(MKLROOT)/include
```

## Limitations of the Direct Call

Directly calling the Intel® oneAPI Math Kernel Library kernels has the following limitations:

- If the `MKL_DIRECT_CALL` or `MKL_DIRECT_CALL_SEQ`macro is used, Intel® oneAPI Math Kernel Library may skip error checking.

  **Important**
  With a limited error checking, you are responsible for checking the correctness of function parameters to avoid unsafe and incorrect code execution.

- The feature is only available for the following functions:
  - BLAS: `?gemm`, `?gemm3m`, `?syrk`, `?trsm`, `?axpy`, and `?dot`
  - LAPACK: `?getrf`, `?getrs`, `?getri`, `?potrf`, and `?geqrf`. (available for C applications only)
- Intel® oneAPI Math Kernel Library Verbose mode, Conditional Numerical Reproducibility, and BLAS95 interfaces are not supported.
- GNU* Fortran compilers are not supported.
- For C applications, you must enable mixing declarations and user code by providing the `-std=c99` option for Intel® compilers.

# Other Tips and Techniques to Improve Performance

## Coding Techniques

To improve performance, properly align arrays in your code. Additional conditions can improve performance for specific function domains.

## Data Alignment and Leading Dimensions

To improve performance of your application that calls Intel® oneAPI Math Kernel Library, align your arrays on 64-byte boundaries and ensure that the leading dimensions of the arrays are divisible by 64/*element_size*, where *element_size* is the number of bytes for the matrix elements (4 for single-precision real, 8 for double-precision real and single-precision complex, and 16 for double-precision complex) . For more details, see Example of Data Alignment.

## LAPACK Packed Routines

The routines with the names that contain the letters `HP, OP, PP, SP, TP, UP`in the matrix type and storage position (the second and third letters respectively) operate on the matrices in the packed format (see LAPACK "Routine Naming Conventions" sections in the Intel® oneAPI Math Kernel Library Developer Reference). Their functionality is strictly equivalent to the functionality of the unpacked routines with the names containing the letters`HE, OR, PO, SY, TR, UN` in the same positions, but the performance is significantly lower.

If the memory restriction is not too tight, use an unpacked routine for better performance. In this case, you need to allocate $N^2/2$ more memory than the memory required by a respective packed routine, where $N$ is the problem size (the number of equations).

For example, to speed up solving a symmetric eigenproblem with an expert driver, use the unpacked routine:

```
call dsyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz, work, lwork,
iwork, ifail, info)
```

where `a` is the dimension *`lda`*-by-*`n`*, which is at least $N^2$ elements,
instead of the packed routine:

```
call dspevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z, ldz, work, iwork, ifail,
info)
```

where `ap` is the dimension $N*(N+1)/2$.

## See Also

## Improving oneMKL Performance on Specific Processors

### Dual-Core Intel® Xeon® Processor 5100 Series

To get the best performance with Intel® oneAPI Math Kernel Library on Dual-Core Intel® Xeon® processor 5100 series systems, enable the Hardware DPL (streaming data) Prefetcher functionality of this processor. To configure this functionality, use the appropriate BIOS settings, as described in your BIOS documentation.

## Operating on Denormals

The IEEE 754-2008 standard, "An IEEE Standard for Binary Floating-Point Arithmetic", defines *denormal* (or *subnormal*) numbers as non-zero numbers smaller than the smallest possible normalized numbers for a specific floating-point format. Floating-point operations on denormals are slower than on normalized operands because denormal operands and results are usually handled through a software assist mechanism rather than directly in hardware. This software processing causes Intel® oneAPI Math Kernel Library functions that consume denormals to run slower than with normalized floating-point numbers.

You can mitigate this performance issue by setting the appropriate bit fields in the MXCSR floating-point control register to flush denormals to zero (FTZ) or to replace any denormals loaded from memory with zero (DAZ). Check your compiler documentation to determine whether it has options to control FTZ and DAZ. Note that these compiler options may slightly affect accuracy.

## Using Memory Functions

### Avoiding Memory Leaks in oneMKL

When running, Intel® oneAPI Math Kernel Library (oneMKL) may allocate and deallocate internal buffers to facilitate better performance. Memory leaks can occur if the Intel® oneAPI Math Kernel Library is unloaded before freeing the internal buffers.

You can free the internal buffers by calling the `mkl_free_buffers()` function or, for more granular control, the `mkl_thread_free_buffers()` function.

Alternatively, setting the `MKL_DISABLE_FAST_MM` environment variable to `1` or calling the `mkl_disable_fast_mm()` function disables the internal memory manager. Be aware that this change may negatively impact the performance of some oneMKL functions, especially for small problem sizes.

#### See Also

Intel Software Documentation Library

### Redefining Memory Functions

In C/C++ programs, you can replace Intel® oneAPI Math Kernel Library memory functions that the library uses by default with your own functions. To do this, use the*memory renaming* feature.

### Memory Renaming

Intel® oneAPI Math Kernel Library memory management by default uses standard C run-time memory functions to allocate or free memory. These functions can be replaced using memory renaming.

Intel® oneAPI Math Kernel Library accesses the memory functions by pointers`i_malloc, i_free, i_calloc`, and `i_realloc`, which are visible at the application level. These pointers initially hold addresses of the standard C run-time memory functions `malloc, free, calloc`, and `realloc`, respectively. You can programmatically redefine values of these pointers to the addresses of your application's memory management functions.

Redirecting the pointers is the only correct way to use your own set of memory management functions. If you call your own memory functions without redirecting the pointers, the memory will get managed by two independent memory management packages, which may cause unexpected memory issues.

### How to Redefine Memory Functions

To redefine memory functions, use the following procedure:

1. Include the `i_malloc.h` header file in your code.
   This header file contains all declarations required for replacing the memory allocation functions. The header file also describes how memory allocation can be replaced in those Intel libraries that support this feature.
2. Redefine values of pointers `i_malloc, i_free, i_calloc`, and `i_realloc`prior to the first call to Intel® oneAPI Math Kernel Library functions, as shown in the following example:

```
#include "i_malloc.h"
  . . .
```

```
              i_malloc  = my_malloc;
              i_calloc  = my_calloc;
              i_realloc = my_realloc;
              i_free    = my_free;
              . . .
            // Now you may call Intel MKL functions
```

# Language-specific Usage Options

5

The Intel® oneAPI Math Kernel Library (oneMKL) provides broad support for Fortran and C/C++ programming. However, not all functions support both Fortran and C interfaces. For example, some LAPACK functions have no C interface. You can call such functions from C using mixed-language programming.

If you want to use LAPACK or BLAS functions that support Fortran 77 in the Fortran 95 environment, additional effort may be initially required to build compiler-specific interface libraries and modules from the source code provided with Intel® oneAPI Math Kernel Library.

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.<br><br>Notice revision #20201201 |

**See Also**

Language Interfaces Support, by Function Domain

## Using Language-Specific Interfaces with Intel® oneAPI Math Kernel Library

This section discusses mixed-language programming and the use of language-specific interfaces with Intel® oneAPI Math Kernel Library.

See also "FFTW Interface to Intel® oneAPI Math Kernel Library" in the Intel® oneAPI Math Kernel Library Developer Reference (available in the Intel Software Documentation Library) for details of the FFTW interfaces to Intel® oneAPI Math Kernel Library.

### Interface Libraries and Modules

You can create the following interface libraries and modules using the respective makefiles located in the interfaces directory.

| File name | Contains |
|---|---|
| **Libraries, in Intel® oneAPI Math Kernel Library architecture-specific directories** | |
| `libmkl_blas95_ilp64.a`[1] | Fortran 95 wrappers for BLAS (BLAS95) supporting LP64 interface. |
| `libmkl_blas95_lp64.a`[1] | Fortran 95 wrappers for BLAS (BLAS95) supporting ILP64 interface. |
| `libmkl_lapack95_lp64.a`[1] | Fortran 95 wrappers for LAPACK (LAPACK95) supporting LP64 interface. |
| `libmkl_lapack95_ilp64.a`[1] | Fortran 95 wrappers for LAPACK (LAPACK95) supporting ILP64 interface. |
| `libfftw2xc_intel.a`[1] | Interfaces for FFTW version 2.x (C interface for Intel compilers) to call Intel® oneAPI Math Kernel Library FFT. |

| File name | Contains |
|---|---|
| `libfftw2xc_gnu.a` | Interfaces for FFTW version 2.x (C interface for GNU compilers) to call Intel® oneAPI Math Kernel Library FFT. |
| `libfftw2xf_intel.a` | Interfaces for FFTW version 2.x (Fortran interface for Intel compilers) to call Intel® oneAPI Math Kernel Library FFT. |
| `libfftw2xf_gnu.a` | Interfaces for FFTW version 2.x (Fortran interface for GNU compiler) to call Intel® oneAPI Math Kernel Library FFT. |
| `libfftw3xc_intel.a`[2] | Interfaces for FFTW version 3.x (C interface for Intel compiler) to call Intel® oneAPI Math Kernel Library FFT. |
| `libfftw3xc_gnu.a` | Interfaces for FFTW version 3.x (C interface for GNU compilers) to call Intel® oneAPI Math Kernel Library FFT. |
| `libfftw3xf_intel.a`[2] | Interfaces for FFTW version 3.x (Fortran interface for Intel compilers) to call Intel® oneAPI Math Kernel Library FFT. |
| `libfftw3xf_gnu.a` | Interfaces for FFTW version 3.x (Fortran interface for GNU compilers) to call Intel® oneAPI Math Kernel Library FFT. |
| **Modules, in architecture- and interface-specific subdirectories of the Intel® oneAPI Math Kernel Library include directory** | |
| `blas95.mod`[1] | Fortran 95 interface module for BLAS (BLAS95). |
| `lapack95.mod`[1] | Fortran 95 interface module for LAPACK (LAPACK95). |
| `f95_precision.mod`[1] | Fortran 95 definition of precision parameters for BLAS95 and LAPACK95. |
| `mkl_service.mod`[1] | Fortran 95 interface module for Intel® oneAPI Math Kernel Library support functions. |

[1] Prebuilt for the Intel® Fortran compiler

[2]FFTW3 interfaces are integrated with Intel® oneAPI Math Kernel Library. Look into`<*mkl directory*>/interfaces/fftw3x*/makefile` for options defining how to build and where to place the standalone library with the wrappers.

## See Also

Fortran 95 Interfaces to LAPACK and BLAS

## Fortran 95 Interfaces to LAPACK and BLAS

Fortran 95 interfaces are compiler-dependent. Intel® oneAPI Math Kernel Library provides the interface libraries and modules precompiled with the Intel® Fortran compiler. Additionally, the Fortran 95 interfaces and wrappers are delivered as sources. (For more information, seeCompiler-dependent Functions and Fortran 90 Modules). If you are using a different compiler, build the appropriate library and modules with your compiler and link the library as a user's library:

1. Go to the respective directory `<*mkl directory*>/interfaces/blas95` or `<*mkl directory*>/interfaces/lapack95`
2. Type:
   - `make libintel64 [interface=lp64|ilp64] INSTALL_DIR=<*user dir*>`

**Important**
The parameter `INSTALL_DIR` is required.

As a result, the required library is built and installed in the `<user dir>/lib` directory, and the `.mod` files are built and installed in the `<user dir>/include/<arch>/[/{lp64|ilp64}]` directory, where `<arch>`is {intel64}.

By default, the ifort compiler is assumed. You may change the compiler with an additional parameter of `make`:
`FC=<compiler>`.

For example, the command

```
make libintel64 FC=pgf95 INSTALL_DIR=<userpgf95 dir> interface=lp64
```

builds the required library and `.mod` files and installs them in subdirectories of `<userpgf95 dir>`.

To delete the library from the building directory, type:

- `make cleanintel64 [interface=lp64|ilp64] INSTALL_DIR=<user dir>`
- `make clean INSTALL_DIR=<user_dir>`

**Caution**
Even if you have administrative rights, avoid setting `INSTALL_DIR=../..` or `INSTALL_DIR=<mkl directory>` in a build or clean command above because these settings replace or delete the Intel® oneAPI Math Kernel Library prebuilt Fortran 95 library and modules.

## Compiler-dependent Functions and Fortran 90 Modules

Compiler-dependent functions occur whenever the compiler inserts into the object code function calls that are resolved in its run-time library (RTL). Linking of such code without the appropriate RTL will result in undefined symbols. Intel® oneAPI Math Kernel Library has been designed to minimize RTL dependencies.

In cases where RTL dependencies might arise, the functions are delivered as source code and you need to compile the code with whatever compiler you are using for your application.

In particular, Fortran 90 modules result in the compiler-specific code generation requiring RTL support. Therefore, Intel® oneAPI Math Kernel Library delivers these modules compiled with the Intel compiler, along with source code, to be used with different compilers.

# Mixed-language Programming with the Intel Math Kernel Library

Appendix A Intel® oneAPI Math Kernel Library Language Interfaces Supportlists the programming languages supported for each Intel® oneAPI Math Kernel Library function domain. However, you can call Intel® oneAPI Math Kernel Library routines from different language environments.

See also these Knowledge Base articles:

- https://software.intel.com/content/www/us/en/develop/articles/how-to-use-boost-ublas-with-intel-mkl.html for how to perform BLAS matrix-matrix multiplication in C++ using Intel® oneAPI Math Kernel Library substitution of Boost* uBLAS functions.
- https://software.intel.com/content/www/us/en/develop/articles/intel-mkl-and-third-party-applications-how-to-use-them-together.html for a list of articles describing how to use Intel® oneAPI Math Kernel Library with third-party libraries and applications.

## Calling LAPACK, BLAS, and CBLAS Routines from C/C++ Language Environments

Not all Intel® oneAPI Math Kernel Library function domains support both C and Fortran environments. To use Intel® oneAPI Math Kernel Library Fortran-style functions in C/C++ environments, you should observe certain conventions, which are discussed for LAPACK and BLAS in the subsections below.

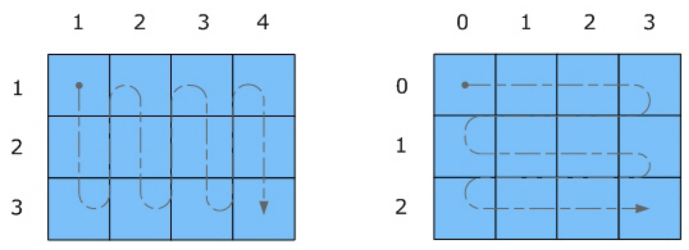
**Caution**
Avoid calling BLAS 95/LAPACK 95 from C/C++. Such calls require skills in manipulating the descriptor of a deferred-shape array, which is the Fortran 90 type. Moreover, BLAS95/LAPACK95 routines contain links to a Fortran RTL.

## LAPACK and BLAS

Because LAPACK and BLAS routines are Fortran-style, when calling them from C-language programs, follow the Fortran-style calling conventions:

- Pass variables by *address*, not by *value*.
  Function calls in Example "Calling a Complex BLAS Level 1 Function from C++" and Example "Using CBLAS Interface Instead of Calling BLAS Directly from C" illustrate this.
- Store your data in Fortran style, that is, column-major rather than row-major order.

With row-major order, adopted in C, the last array index changes most quickly and the first one changes most slowly when traversing the memory segment where the array is stored. With Fortran-style column-major order, the last index changes most slowly whereas the first index changes most quickly (as illustrated by the figure below for a two-dimensional array).



A: Column-major order (Fortran-style)    B: Row-major order (C-style)

For example, if a two-dimensional matrix A of size `mxn` is stored densely in a one-dimensional array B, you can access a matrix element like this:

`A[i][j] = B[i*n+j]` in C        ( `i=0, ... , m-1, j=0, ... , -1`)

`A(i,j)  = B((j-1)*m+i)` in Fortran ( `i=1, ... , m, j=1, ... , n`).

When calling LAPACK or BLAS routines from C, be aware that because the Fortran language is case-insensitive, the routine names can be both upper-case or lower-case, with or without the trailing underscore. For example, the following names are equivalent:

- LAPACK: `dgetrf`, `DGETRF`, `dgetrf_`, and `DGETRF_`
- BLAS: `dgemm`, `DGEMM`, `dgemm_`, and `DGEMM_`

See Example "Calling a Complex BLAS Level 1 Function from C++" on how to call BLAS routines from C.

See also the Intel® oneAPI Math Kernel Library Developer Reference for a description of the C interface to LAPACK functions.

## CBLAS

Instead of calling BLAS routines from a C-language program, you can use the CBLAS interface.

CBLAS is a C-style interface to the BLAS routines. You can call CBLAS routines using regular C-style calls. Use the `mkl.h` header file with the CBLAS interface. `mkl.h` includes the `mkl_cblas.h` header file, which specifies enumerated values and prototypes of all the functions. It also determines whether the program is being compiled with a C++ compiler, and if it is, the included file will be correct for use with C++ compilation. Example "Using CBLAS Interface Instead of Calling BLAS Directly from C" illustrates the use of the CBLAS interface.

## C Interface to LAPACK

Instead of calling LAPACK routines from a C-language program, you can use the C interface to LAPACK provided by Intel® oneAPI Math Kernel Library.

The C interface to LAPACK is a C-style interface to the LAPACK routines. This interface supports matrices in row-major and column-major order, which you can define in the first function argument *matrix_order*. Use the `mkl.h` header file with the C interface to LAPACK. `mkl.h` includes the `mkl_lapacke.h` header file, which specifies constants and prototypes of all the functions. It also determines whether the program is being compiled with a C++ compiler, and if it is, the included file will be correct for use with C++ compilation. You can find examples of the C interface to LAPACK in the `examples/lapacke`subdirectory in the Intel® oneAPI Math Kernel Library installation directory.

## Using Complex Types in C/C++

As described in the documentation for the Intel® Fortran Compiler, C/C++ does not directly implement the Fortran types `COMPLEX(4)` and `COMPLEX(8)`. However, you can write equivalent structures. The type `COMPLEX(4)` consists of two 4-byte floating-point numbers. The first of them is the real-number component, and the second one is the imaginary-number component. The type `COMPLEX(8)` is similar to `COMPLEX(4)` except that it contains two 8-byte floating-point numbers.

Intel® oneAPI Math Kernel Library provides complex types`MKL_Complex8` and `MKL_Complex16`, which are structures equivalent to the Fortran complex types `COMPLEX(4)` and `COMPLEX(8)`, respectively. The `MKL_Complex8` and `MKL_Complex16` types are defined in the `mkl_types.h` header file. You can use these types to define complex data. You can also redefine the types with your own types before including the `mkl_types.h` header file. The only requirement is that the types must be compatible with the Fortran complex layout, that is, the complex type must be a pair of real numbers for the values of real and imaginary parts.

For example, you can use the following definitions in your C++ code:

```
#define MKL_Complex8 std::complex<float>
```

and

```
#define MKL_Complex16 std::complex<double>
```

See Example "Calling a Complex BLAS Level 1 Function from C++" for details. You can also define these types in the command line:

```
-DMKL_Complex8="std::complex<float>"
-DMKL_Complex16="std::complex<double>"
```

### See Also

Intel® Software Documentation Library  for the Intel® Fortran Compiler documentation
for the Intel® Fortran Compiler documentation

## Calling BLAS Functions that Return the Complex Values in C/C++ Code

Complex values that functions return are handled differently in C and Fortran. Because BLAS is Fortran-style, you need to be careful when handling a call from C to a BLAS function that returns complex values. However, in addition to normal function calls, Fortran enables calling functions as though they were subroutines, which provides a mechanism for returning the complex value correctly when the function is called from a C program. When a Fortran function is called as a subroutine, the return value is the first parameter in the calling sequence. You can use this feature to call a BLAS function from C.

The following example shows how a call to a Fortran function as a subroutine converts to a call from C and the hidden parameter result gets exposed:

Normal Fortran function call: `result = cdotc( n, x, 1, y, 1 )`

A call to the function as a subroutine: `call cdotc( result, n, x, 1, y, 1)`

A call to the function from C: `cdotc( &result, &n, x, &one, y, &one )`

**NOTE**
Intel® oneAPI Math Kernel Library has both upper-case and lower-case entry points in the Fortran-style (case-insensitive) BLAS, with or without the trailing underscore. So, all these names are equivalent and acceptable:`cdotc`, `CDOTC`, `cdotc_`, and `CDOTC_`.

The above example shows one of the ways to call several level 1 BLAS functions that return complex values from your C and C++ applications. An easier way is to use the CBLAS interface. For instance, you can call the same function using the CBLAS interface as follows:

`cblas_cdotc( n, x, 1, y, 1, &result )`

**NOTE**
The complex value comes last on the argument list in this case.

The following examples show use of the Fortran-style BLAS interface from C and C++, as well as the CBLAS (C language) interface:

- Example "Calling a Complex BLAS Level 1 Function from C"
- Example "Calling a Complex BLAS Level 1 Function from C++"
- Example "Using CBLAS Interface Instead of Calling BLAS Directly from C"

## Example "Calling a Complex BLAS Level 1 Function from C"

The example below illustrates a call from a C program to the complex BLAS Level 1 function `zdotc()`. This function computes the dot product of two double-precision complex vectors.

In this example, the complex dot product is returned in the structure `c`.

```
#include "mkl.h"
#define N 5
int main()
{
int n = N, inca = 1, incb = 1, i;
MKL_Complex16 a[N], b[N], c;
for( i = 0; i < n; i++ )
{
 a[i].real = (double)i; a[i].imag = (double)i * 2.0;
 b[i].real = (double)(n - i); b[i].imag = (double)i * 2.0;
}
zdotc( &c, &n, a, &inca, b, &incb );
printf( "The complex dot product is: ( %6.2f, %6.2f)\n", c.real, c.imag );
return 0;
}
```

In this example, the complex dot product for large data size is returned in the structure `c`.

```
#include "mkl.h"
 #define N 5
 int main()
 {
     MKL_INT64 n = N, inca = 1, incb = 1, i;
     MKL_Complex16 a[N], b[N], c;
     for( i = 0; i < n; i++ )
     {
         a[i].real = (double)i; a[i].imag = (double)i * 2.0;
         b[i].real = (double)(n - i); b[i].imag = (double)i * 2.0;
     }
     zdotc_64( &c, &n, a, &inca, b, &incb );
     printf( "The complex dot product is: ( %6.2f, %6.2f)\n", c.real, c.imag );
     return 0;
 }
```

## Example "Calling a Complex BLAS Level 1 Function from C++"

Below is the C++ implementation:

```
#include <complex>
#include <iostream>
#define MKL_Complex16 std::complex<double>
#include "mkl.h"

#define N 5

int main()
{
    int n, inca = 1, incb = 1, i;
    std::complex<double> a[N], b[N], c;
    n = N;

    for( i = 0; i < n; i++ )
 {
        a[i] = std::complex<double>(i,i*2.0);
        b[i] = std::complex<double>(n-i,i*2.0);
    }
    zdotc(&c, &n, a, &inca, b, &incb );
    std::cout << "The complex dot product is: " << c << std::endl;
    return 0;
}
```

## Example "Using CBLAS Interface Instead of Calling BLAS Directly from C"

This example uses CBLAS:

```
#include <stdio.h>
#include "mkl.h"
typedef struct{ double re; double im; } complex16;
#define N 5
int main()
{
int n, inca = 1, incb = 1, i;
complex16 a[N], b[N], c;
n = N;
for( i = 0; i < n; i++ )
{
 a[i].re = (double)i; a[i].im = (double)i * 2.0;
 b[i].re = (double)(n - i); b[i].im = (double)i * 2.0;
}
cblas_zdotc_sub(n, a, inca, b, incb, &c );
printf( "The complex dot product is: ( %6.2f, %6.2f)\n", c.re, c.im );
return 0;
}
```

# Obtaining Numerically Reproducible Results

# 6

Intel® oneAPI Math Kernel Library (oneMKL) offers functions and environment variables that help you obtain Conditional Numerical Reproducibility (CNR) of floating-point results when calling the library functions from your application. These new controls enable Intel® oneAPI Math Kernel Library to run in a special mode, when functions return bitwise reproducible floating-point results from run to run under the following conditions:

- Calls to Intel® oneAPI Math Kernel Library occur in a single executable
- The number of computational threads used by the library does not change in the run

For a limited set of routines, you can eliminate the second condition by using Intel® oneAPI Math Kernel Library in strict CNR mode.

It is well known that for general single and double precision IEEE floating-point numbers, the associative property does not always hold, meaning (a+b)+c may not equal a +(b+c). Let's consider a specific example. In infinite precision arithmetic $2^{-63} + 1 + -1 = 2^{-63}$. If this same computation is done on a computer using double precision floating-point numbers, a rounding error is introduced, and the order of operations becomes important:

$(2^{-63} + 1) + (-1) \simeq 1 + (-1) = 0$

versus

$2^{-63} + (1 + (-1)) \simeq 2^{-63} + 0 = 2^{-63}$

This inconsistency in results due to order of operations is precisely what the new functionality addresses.

The application related factors that affect the order of floating-point operations within a single executable program include selection of a code path based on run-time processor dispatching, alignment of data arrays, variation in number of threads, threaded algorithms and internal floating-point control settings. You can control most of these factors by controlling the number of threads and floating-point settings and by taking steps to align memory when it is allocated (see the Getting Reproducible Results with Intel® MKL knowledge base article for details). However, run-time dispatching and certain threaded algorithms do not allow users to make changes that can ensure the same order of operations from run to run.

Intel® oneAPI Math Kernel Library does run-time processor dispatching in order to identify the appropriate internal code paths to traverse for the Intel® oneAPI Math Kernel Library functions called by the application. The code paths chosen may differ across a wide range of Intel processors and Intel architecture compatible processors and may provide differing levels of performance. For example, an Intel® oneAPI Math Kernel Library function running on an Intel® Pentium® 4 processor may run one code path, while on the latest Intel® Xeon® processor it will run another code path. This happens because each unique code path has been optimized to match the features available on the underlying processor. One key way that the new features of a processor are exposed to the programmer is through the instruction set architecture (ISA). Because of this, code branches in Intel® oneAPI Math Kernel Library are designated by the latest ISA they use for optimizations: from the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) to the Intel® Advanced Vector Extensions2 (Intel® AVX2). The feature-based approach introduces a challenge: if any of the internal floating-point operations are done in a different order or are re-associated, the computed results may differ.

Dispatching optimized code paths based on the capabilities of the processor on which the code is running is central to the optimization approach used by Intel® oneAPI Math Kernel Library. So it is natural that consistent results require some performance trade-offs. If limited to a particular code path, performance of Intel® oneAPI Math Kernel Library can in some circumstances degrade by more than a half. To understand this, note that matrix-multiply performance nearly doubled with the introduction of new processors supporting Intel AVX2 instructions. Even if the code branch is not restricted, performance can degrade by 10-20% because the new functionality restricts algorithms to maintain the order of operations.

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. |

| **Product and Performance Information** |
|---|
| Notice revision #20201201 |

# Getting Started with Conditional Numerical Reproducibility

Intel® oneAPI Math Kernel Library offers functions and environment variables to help you get reproducible results. You can configure Intel® oneAPI Math Kernel Library using functions or environment variables, but the functions provide more flexibility.

The following specific examples introduce you to the conditional numerical reproducibility.

While these examples recommend aligning input and output data, you can supply unaligned data to Intel® oneAPI Math Kernel Library functions running in the CNR mode, but refer toReproducibility Conditions for details related to data alignment.

## Intel CPUs supporting Intel AVX2

To ensure Intel® oneAPI Math Kernel Library calls return the same results on every Intel CPU supporting Intel AVX2 instructions:

1. Make sure that your application uses a fixed number of threads
2. (Recommended) Properly align input and output arrays in Intel® oneAPI Math Kernel Library function calls
3. Do either of the following:
   - Call

     `mkl_cbwr_set(MKL_CBWR_AVX2)`
   - Set the environment variable:

     `export MKL_CBWR = AVX2`

**NOTE**
On non-Intel CPUs and on Intel CPUs that do not support Intel AVX2, this environment setting may cause results to differ because the `AUTO` branch is used instead, while the above function call returns an error and does not enable the CNR mode.

## Intel CPUs supporting Intel SSE2

To ensure Intel® oneAPI Math Kernel Library calls return the same results on every Intel CPU supporting Intel SSE2instructions:

1. Make sure that your application uses a fixed number of threads
2. (Recommended) Properly align input and output arrays in Intel® oneAPI Math Kernel Library function calls
3. Do either of the following:
   - Call

     `mkl_cbwr_set(MKL_CBWR_SSE2)`
   - Set the environment variable:

     `export MKL_CBWR = SSE2`

**NOTE**
On non-Intel CPUs, this environment setting may cause results to differ because the `AUTO` branch is used instead, while the above function call returns an error and does not enable the CNR mode.

### Intel or Intel compatible CPUs supporting Intel SSE2

On non-Intel CPUs, only the `MKL_CBWR_AUTO` and `MKL_CBWR_COMPATIBLE` options are supported for function calls and only `AUTO` and `COMPATIBLE` options for environment settings.

To ensure Intel® oneAPI Math Kernel Library calls return the same results on all Intel or Intel compatible CPUs supporting Intel SSE2 instructions:

1. Make sure that your application uses a fixed number of threads
2. (Recommended) Properly align input and output arrays in Intel® oneAPI Math Kernel Library function calls
3. Do either of the following:
   - Call

     ```
     mkl_cbwr_set(MKL_CBWR_COMPATIBLE)
     ```
   - Set the environment variable:

     ```
     export MKL_CBWR = COMPATIBLE
     ```

**NOTE**
The special `MKL_CBWR_COMPATIBLE/COMPATIBLE`option is provided because Intel and Intel compatible CPUs have a few instructions, such as approximation instructions rcpps/rsqrtps, that may return different results. This option ensures that Intel® oneAPI Math Kernel Library does not use these instructions and forces a single Intel SSE2 only code path to be executed.

### Next steps

| | |
|---|---|
| See Specifying the Code Branches | for details of specifying the branch using environment variables. |

See the following sections in the *Intel® oneAPI Math Kernel Library Developer Reference*:

| | |
|---|---|
| Support Functions for Conditional Numerical Reproducibility | for how to configure the CNR mode of Intel® oneAPI Math Kernel Library using functions. |
| Intel® oneAPI Math Kernel Library PARDISO - Parallel Direct Sparse Solver Interface | for how to configure the CNR mode for PARDISO. |

### See Also
Code Examples

## Specifying Code Branches

Intel® oneAPI Math Kernel Library provides a conditional numerical reproducibility (CNR) functionality that enables you to obtain reproducible results from oneMKL routines. When enabling CNR, you choose a specific code branch of Intel® oneAPI Math Kernel Library that corresponds to the instruction set architecture (ISA) that you target. You can specify the code branch and other CNR options using the`MKL_CBWR` environment variable.

- `MKL_CBWR="`*`<branch>[,STRICT]`*`"` or
- `MKL_CBWR="BRANCH=`*`<branch>[,STRICT]`*`"`

Use the `STRICT` flag to enable strict CNR mode. For more information, see Reproducibility Conditions.

The *`<branch>`* placeholder specifies the CNR branch with one of the following values:

| **Value** | **Description** |
|---|---|
| `AUTO` | CNR mode uses the standard ISA-based dispatching model while ensuring fixed cache sizes, deterministic reductions, and static scheduling |
| `COMPATIBLE` | Intel® Streaming SIMD Extensions 2 (Intel® SSE2) without rcpps/ rsqrtps instructions |
| `SSE4_2` | Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) |
| `AVX` | Intel® Advanced Vector Extensions (Intel® AVX) |
| `AVX2` | Intel® Advanced Vector Extensions 2 (Intel® AVX2) |
| `AVX512` | Intel AVX-512 on Intel® Xeon® processors |
| `AVX512_E1` | Intel® Advanced Vector Extensions 512 (Intel® AVX-512) with support for Vector Neural Network Instructions |
| `AVX512_MIC` | DEPRECATED. Intel® Advanced Vector Extensions 512 (Intel® AVX-512) on Intel® Xeon Phi™ processors. This setting is kept for backward compatibility and is equivalent to `AVX2`. |
| `AVX512_MIC_E1` | DEPRECATED. Intel® Advanced Vector Extensions 512 (Intel® AVX-512) with support for Vector Neural Network Instructions on Intel® Xeon Phi™ processors. This setting is kept for backward compatibility and is equivalent to `AVX2`. |

When specifying the CNR branch, be aware of the following:

- Reproducible results are provided under Reproducibility Conditions.
- Settings other than `AUTO` or `COMPATIBLE` are available only for Intel processors.
- To get the CNR branch optimized for the processor where your program is currently running, choose the value of `AUTO` or call the `mkl_cbwr_get_auto_branch` function.
- Strict CNR mode is supported only for AVX2, AVX512, AVX512_E1, AVX512_MIC, and AVX512_MIC_E1 branches. You can also use strict CNR mode with the AUTO branch when running on Intel processors that support one of these instruction set architectures (ISAs).

Setting the `MKL_CBWR` environment variable or a call to an equivalent `mkl_cbwr_set` function fixes the code branch and sets the reproducibility mode.

**NOTE**

- If the value of the branch is incorrect or your processor or operating system does not support the specified ISA, CNR ignores this value and uses the `AUTO` branch without providing any warning messages.
- Calls to functions that define the behavior of CNR must precede any of the math library functions that they control.
- Settings specified by the functions take precedence over the settings specified by the environment variable.

See the *Intel® oneAPI Math Kernel Library Developer Reference* for how to specify the branches using functions.

**See Also**
Getting Started with Conditional Numerical Reproducibility

## Reproducibility Conditions

To get reproducible results from run to run, ensure that the number of threads is fixed and constant. Specifically:

- If you are running your program with OpenMP* parallelization on different processors, explicitly specify the number of threads.
- To ensure that your application has deterministic behavior with OpenMP* parallelization and does not adjust the number of threads dynamically at run time, set `MKL_DYNAMIC` and `OMP_DYNAMIC` to FALSE. This is especially needed if you are running your program on different systems.
- If you are running your program with the Intel® Threading Building Blocks parallelization, numerical reproducibility is not guaranteed.

### Strict CNR Mode

In strict CNR mode, oneAPI Math Kernel Library provides bitwise reproducible results for a limited set of functions and code branches even when the number of threads changes. These routines and branches support strict CNR mode (64-bit libraries only):

- `?gemm`, `?symm`, `?hemm`, `?trsm` and their CBLAS equivalents (`cblas_?gemm`, `cblas_?symm`, `cblas_?hemm`, and `cblas_?trsm`).
- Intel® Advanced Vector Extensions 2 (Intel® AVX2) or Intel® Advanced Vector Extensions 512 (Intel® AVX-512).

When using other routines or CNR branches,oneAPI Math Kernel Library operates in standard (non-strict) CNR mode, subject to the restrictions described above. Enabling strict CNR mode can reduce performance.

**NOTE**

- As usual, you should align your data, even in CNR mode, to obtain the best possible performance. While CNR mode also fully supports unaligned input and output data, the use of it might reduce the performance of some oneAPI Math Kernel Library functions on earlier Intel processors. Refer to coding techniques that improve performance for more details.
- Conditional Numerical Reproducibility does not ensure that bitwise-identical NaN values are generated when the input data contains NaN values.
- If dynamic memory allocation fails on one run but succeeds on another run, you may fail to get reproducible results between these two runs.

**See Also**
MKL_DYNAMIC
Coding Techniques

## Setting the Environment Variable for Conditional Numerical Reproducibility

The following examples illustrate the use of the `MKL_CBWR` environment variable. The first command in each list sets Intel® oneAPI Math Kernel Library to run in the CNR mode based on the default dispatching for your platform. The other two commandsin each list are equivalent and set the CNR branch to Intel AVX.

For the bash shell:

- `export MKL_CBWR="AUTO"`

- `export MKL_CBWR="AVX"`
- `export MKL_CBWR="BRANCH=AVX"`

For the C shell (csh or tcsh):

- `setenv MKL_CBWR "AUTO"`
- `setenv MKL_CBWR "AVX"`
- `setenv MKL_CBWR "BRANCH=AVX"`

**See Also**
Specifying Code Branches

## Code Examples

The following simple programs show how to obtain reproducible results from run to run of Intel® oneAPI Math Kernel Library functions. See the*Intel® oneAPI Math Kernel Library Developer Reference* for more examples.

### C Example of CNR

```
#include <mkl.h>
int main(void) {
    int my_cbwr_branch;
    /* Align all input/output data on 64-byte boundaries */
    /* "for best performance of Intel® oneAPI Math Kernel Library */
    void *darray;
    int darray_size=1000;
    /* Set alignment value in bytes */
    int alignment=64;
    /* Allocate aligned array */
    darray = mkl_malloc (sizeof(double)*darray_size, alignment);
    /* Find the available MKL_CBWR_BRANCH automatically */
    my_cbwr_branch = mkl_cbwr_get_auto_branch();
    /* User code without oneMKL calls */
    /* Piece of the code where CNR of oneMKL is needed */
    /* The performance of oneMKL functions might be reduced for CNR mode */
/* If the "IF" statement below is commented out, Intel® oneAPI Math Kernel Library will run in a
regular mode, */
    /* and data alignment will allow you to get best performance */
    if (mkl_cbwr_set(my_cbwr_branch)) {
        printf("Error in setting MKL_CBWR_BRANCH! Aborting…\n”);
        return;
    }
    /* CNR calls to oneMKL + any other code */
    /* Free the allocated aligned array */
    mkl_free(darray);
}
```

### Fortran Example of CNR

```
    PROGRAM MAIN
    INCLUDE 'mkl.fi'
    INTEGER*4 MY_CBWR_BRANCH
! Align all input/output data on 64-byte  boundaries
! "for best performance of Intel® oneAPI Math Kernel Library
! Declare oneMKL memory allocation routine
#ifdef _IA32
    INTEGER MKL_MALLOC
#else
    INTEGER*8 MKL_MALLOC
```

```
#endif
    EXTERNAL MKL_MALLOC, MKL_FREE
    DOUBLE PRECISION DARRAY
    POINTER (P_DARRAY,DARRAY(1))
    INTEGER DARRAY_SIZE
    PARAMETER (DARRAY_SIZE=1000)
! Set alignment value in bytes
    INTEGER ALIGNMENT
    PARAMETER (ALIGNMENT=64)
! Allocate aligned array
    P_DARRAY = MKL_MALLOC (%VAL(8*DARRAY_SIZE), %VAL(ALIGNMENT));
! Find the available MKL_CBWR_BRANCH automatically
    MY_CBWR_BRANCH = MKL_CBWR_GET_AUTO_BRANCH()
! User code without oneMKL calls
! Piece of the code where CNR of oneMKL is needed
! The performance of oneMKL functions may be reduced for CNR mode
! If the "IF" statement below is commented out, Intel® oneAPI Math Kernel Library will run in a
regular mode,
! and data alignment will allow you to get best performance
    IF (MKL_CBWR_SET (MY_CBWR_BRANCH) .NE. MKL_CBWR_SUCCESS) THEN
        PRINT *, 'Error in setting MKL_CBWR_BRANCH! Aborting…'
        RETURN
    ENDIF
! CNR calls to oneMKL + any other code
! Free the allocated aligned array
    CALL MKL_FREE(P_DARRAY)

    END
```

## Use of CNR with Unaligned Data in C

```
#include <mkl.h>
int main(void) {
     int my_cbwr_branch;
     /* If it is not possible to align all input/output data on 64-byte boundaries */
     /* to achieve performance, use unaligned IO data with possible performance */
     /* penalty */
     /* Using unaligned IO data */
     double *darray;
     int darray_size=1000;
     /* Allocate array, malloc aligns data on 8/16-byte boundary only */
     darray = (double *)malloc (sizeof(double)*darray_size);
     /* Find the available MKL_CBWR_BRANCH automatically */
     my_cbwr_branch = mkl_cbwr_get_auto_branch();
     /* User code without oneMKL calls */
     /* Piece of the code where CNR of oneMKL is needed */
     /* The performance of oneMKL functions might be reduced for CNR mode */
     /* If the "IF" statement below is commented out, oneMKL will run in a regular mode, */
     /* and you will NOT get best performance without data alignment */
     if (mkl_cbwr_set(my_cbwr_branch)) {
          printf("Error in setting MKL_CBWR_BRANCH! Aborting…\n");
          return;
}
     /* CNR calls to oneMKL + any other code */
     /* Free the allocated array */
     free(darray);
```

## Use of CNR with Unaligned Data in Fortran

```
     PROGRAM MAIN
     INCLUDE 'mkl.fi'
     INTEGER*4 MY_CBWR_BRANCH
! If it is not possible to align all input/output data on 64-byte boundaries
! to achieve performance, use unaligned IO data with possible performance
! penalty
     DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: DARRAY
     INTEGER DARRAY_SIZE, STATUS
     PARAMETER (DARRAY_SIZE=1000)
! Allocate array with undefined alignment
     ALLOCATE(DARRAY(DARRAY_SIZE));
! Find the available MKL_CBWR_BRANCH automatically
     MY_CBWR_BRANCH = MKL_CBWR_GET_AUTO_BRANCH()
! User code without oneMKL calls
! Piece of the code where CNR of oneMKL is needed
! The performance of oneMKL functions might be reduced for CNR mode
! If the "IF" statement below is commented out, oneMKL will run in a regular mode,
! and you will NOT get best performance without data alignment
     IF (MKL_CBWR_SET(MY_CBWR_BRANCH) .NE. MKL_CBWR_SUCCESS) THEN
          PRINT *, 'Error in setting MKL_CBWR_BRANCH! Aborting…'
          RETURN
     ENDIF
! CNR calls to oneMKL + any other code
! Free the allocated array
     DEALLOCATE(DARRAY)
     END
```

# 7 Coding Tips

This section provides coding tips for managing data alignment and version-specific compilation.

**See Also**

Mixed-language Programming with the Intel® oneAPI Math Kernel Library Tips on language-specific programming

Managing Performance and Memory Coding tips related to performance improvement and use of memory functions

Obtaining Numerically Reproducible Results Tips for obtaining numerically reproducible results of computations

## Example of Data Alignment

Needs for best performance with Intel® oneAPI Math Kernel Library or for reproducible results from run to run of Intel® oneAPI Math Kernel Library functions require alignment of data arrays. The following example shows how to align an array on 64-byte boundaries. To do this, use`mkl_malloc()` in place of system provided memory allocators, as shown in the code example below.

### Aligning Addresses on 64-byte Boundaries

```
// ******* C language *******
...
#include <stdlib.h>
#include <mkl.h>
...
void *darray;
int workspace;
// Set value of alignment
int alignment=64;
...
// Allocate aligned workspace
darray = mkl_malloc( sizeof(double)*workspace, alignment );
...
// call the program using oneMKL
mkl_app( darray );
...
// Free workspace
mkl_free( darray );
```

```
! ******* Fortran language *******
...
! Set value of alignment
integer    alignment
parameter (alignment=64)
...
! Declare oneMKL routines
#ifdef _IA32
integer mkl_malloc
#else
```

```
            integer*8 mkl_malloc
            #endif
            external mkl_malloc, mkl_free, mkl_app
            ...
            double precision darray
            pointer (p_wrk,darray(1))
            integer workspace
            ...
            ! Allocate aligned workspace
            p_wrk = mkl_malloc( %val(8*workspace), %val(alignment) )
            ...
            ! call the program using oneMKL
            call mkl_app( darray )
            ...
            ! Free workspace
            call mkl_free(p_wrk)
```

## Using Predefined Preprocessor Symbols for Intel® MKL Version-Dependent Compilation

Preprocessor symbols (macros) substitute values in a program before it is compiled. The substitution is performed in the preprocessing phase.

The following preprocessor symbols are available:

| Predefined Preprocessor Symbol | Description |
|---|---|
| `__INTEL_MKL__` | Intel® oneAPI Math Kernel Library major version |
| `__INTEL_MKL_MINOR__` | Intel® oneAPI Math Kernel Library minor version |
| `__INTEL_MKL_UPDATE__` | Intel® oneAPI Math Kernel Library update number |
| `INTEL_MKL_VERSION` | Intel® oneAPI Math Kernel Library full version in the following format: `INTEL_MKL_VERSION = (__INTEL_MKL__*100+__INTEL_MKL_MINOR__)*100+__INTEL_MKL_UPDATE__` |

These symbols enable conditional compilation of code that uses new features introduced in a particular version of the library.

To perform conditional compilation:

1. Depending on your compiler, include in your code the file where the macros are defined:

| | |
|---|---|
| C/C++ compiler: | `mkl_version.h`, or `mkl.h`, which includes `mkl_version.h` |
| Intel®Fortran compiler: | `mkl.fi` |
| Any Fortran compiler with enabled preprocessing: | `mkl_version.h` Read the documentation for your compiler for the option that enables preprocessing. |

2. [Optionally] Use the following preprocessor directives to check whether the macro is defined:
   - `#ifdef, #endif` for C/C++
   - `!DEC$IF DEFINED, !DEC$ENDIF` for Fortran
3. Use preprocessor directives for conditional inclusion of code:

- `#if, #endif` for C/C++
- `!DEC$IF, !DEC$ENDIF` for Fortran

**Example**

This example shows how to compile a code segment conditionally for a specific version of Intel® oneAPI Math Kernel Library. In this case, the version is 11.2 Update 4:

**Intel®Fortran Compiler:**

```
include "mkl.fi"
!DEC$IF DEFINED INTEL_MKL_VERSION
!DEC$IF INTEL_MKL_VERSION .EQ. 110204
*   Code to be conditionally compiled
!DEC$ENDIF
!DEC$ENDIF
```

**C/C++ Compiler. Fortran Compiler with Enabled Preprocessing:**

```
#include "mkl.h"
#ifdef INTEL_MKL_VERSION
#if INTEL_MKL_VERSION ==  110204
...     Code to be conditionally compiled
#endif
#endif
```

# Managing Output 8

## Using oneMKL Verbose Mode

When building applications that call Intel® oneAPI Math Kernel Library functions, it may be useful to determine:

- which computational functions are called
- what parameters are passed to them
- how much time is spent to execute the functions
- (for GPU applications) which GPU device the kernel is executed on

You can get an application to print this information to a standard output device by enabling Intel® oneAPI Math Kernel Library Verbose. Functions that can print this information are referred to as *verbose-enabled functions*.

When Verbose mode is active in an Intel® oneAPI Math Kernel Library domain, every call of a verbose-enabled function finishes with printing a human-readable line describing the call. However, if your application gets terminated for some reason during the function call, no information for that function will be printed. The first call to a verbose-enabled function also prints a version information line.

For GPU applications, additional information (one or more GPU information lines) will also be printed by the first call to a verbose-enabled function, following the version information line which will be printed for the host CPU. If there is more than one GPU detected, each GPU device will be printed in a separate line.

We have different implementations for verbose with CPU applications and verbose with GPU applications. The Intel® MKL Verbose mode has 2 modes when used with CPU applications: disabled (default) and enabled. The Intel® MKL Verbose mode has three modes when used with GPU applications: disabled (default), enabled without timing, and enabled with synchronous timing.

To change the verbose mode, either set the environment variable `MKL_VERBOSE`:

| | **CPU application** | **GPU application** |
|---|---|---|
| Set `MKL_VERBOSE` to 0 | to disable Verbose | to disable Verbose |
| Set `MKL_VERBOSE` to 1 | to enable Verbose | to enable Verbose without timing |
| Set `MKL_VERBOSE` to 2 | to enable Verbose | to enable Verbose with synchronous timing |

*or* call the support function `mkl_verbose(int mode)`:

| | **CPU application** | **GPU application** |
|---|---|---|
| Call `mkl_verbose(0)` | to disable Verbose | to disable Verbose |
| Call `mkl_verbose(1)` | to enable Verbose | to enable Verbose without timing |
| Call `mkl_verbose(2)` | to enable Verbose | to enable Verbose with synchronous timing |

### Verbose with CPU Applications

Verbose output will be consisted of version information line and call description lines for CPU.

For CPU applications, you can enable Intel® oneAPI Math Kernel Library Verbose mode in these domains:

- BLAS (and BLAS-like extensions)
- LAPACK
- ScaLAPACK (selected functionality)
- FFT

## Verbose with GPU Applications

The verbose feature is enabled for GPU applications that uses DPC++ API or C/Fortran API with OpenMP offload. When used with GPU applications, verbose allows the measurement of execution time to be enabled or disabled with verbose mode. Timing is taken synchronously, so if verbose is enabled with timing, kernel executions will become synchronous (previous kernel will block later kernels)

Verbose output will be consisted of version information line, GPU information lines, and call description lines for GPU.

**NOTE** Timing for GPU applications is reported for overall execution. For selected functionality device execution time can be also reported if the input queue was created with profiling information.

For GPU applications, you can enable Intel® oneAPI Math Kernel Library Verbose mode in these domains:

- BLAS (and BLAS-like extensions)
- LAPACK
- FFT

## For Both CPU and GPU Verbose

Both enabling and disabling of the Verbose mode using the function call takes precedence over the environment setting. For a full description of the mkl_verbose function, see either the Intel® oneAPI Math Kernel Library Developer Reference for C or the Intel® oneAPI Math Kernel Library Developer Reference for Fortran. Both references are available in the Intel® Software Documentation Library.

Intel® oneAPI Math Kernel Library Verbose mode is not a thread-local but a global state. In other words, if an application changes the mode from multiple threads, the result is undefined.

**WARNING**
The performance of an application may degrade with the Verbose mode enabled, especially when the number of calls to verbose-enabled functions is large, because every call to a verbose-enabled function requires an output operation.

### See Also
Intel Software Documentation Library

## Version Information Line

In the Intel® oneAPI Math Kernel Library Verbose mode, the first call to a verbose-enabled function prints a version information line. The line begins with the`MKL_VERBOSE` character string and uses spaces as delimiters. The format of the rest of the line may change in a future release.

The following table lists information contained in a version information line and provides available links for more information:

| Information | Description | Related Links |
|---|---|---|
| Intel® oneAPI Math Kernel Library version. | This information is separated by a comma from the rest of the line. | |
| Operating system. | Possible values:<br>• `Lnx` for Linux* OS<br>• `Win` for Windows* OS<br>• `OSX` for macOS* | |

| Information | Description | Related Links |
|---|---|---|
| The host CPU frequency. | | |
| Intel® oneAPI Math Kernel Library interface layer used by the application. | Possible values: <br>• <br>• `lp64` or `ilp64` on systems based on the Intel® 64 architecture. | Using the ILP64 Interface vs. LP64 Interface |
| Intel® oneAPI Math Kernel Library threading layer used by the application. | Possible values: <br>`intel_thread`, `tbb_thread`, or `sequential`. | Linking with Threading Libraries |

The following is an example of a version information line:

```
MKL_VERBOSE Intel(R) MKL 11.2 Beta build 20131126 for Intel(R) 64 architecture Intel(R)
Advanced Vector Extensions (Intel(R) AVX) Enabled Processor, OSX 3.10GHz lp64
intel_thread
```

## GPU Information Line

In Intel® oneAPI Math Kernel Library Verbose mode for GPU applications, the first call to a verbose-enabled function prints out the GPU information line or lines for all detected GPU devices, each in a separate line. The line begins with the `MKL_VERBOSE Detected` character string and uses spaces as delimiters. The format of the rest of the line may change in a future release.

Only Intel® GPU is supported.

The following table lists information contained in a GPU information line.

**See Also**Call Description Line for GPU

| Information | Description |
|---|---|
| GPU index | The index of the GPU device will be printed after the character string "GPU" (e.g. GPU0, GPU1, GPU2, etc). This GPU index will be used as a nickname of the device in call description lines to refer to the device. |
| Intel® GPU architecture | The value can be one of the following: <br>• Intel(R) Gen9 <br>• Intel(R) Xe_LP <br>• Intel(R) Xe_HP <br>• Intel(R) Xe_HPG <br>• Intel(R) Xe_HPC <br>• Unknown GPU |
| Runtime backend | The value printed is prefixed with Backend: |
| Vector Engine number | The value printed is prefixed with VE: |
| Stack number | The value printed is prefixed with Stack: |
| Maximum workgroup size | The value printed is prefixed with maxWGsize: |

The following is an example of a GPU information line:

```
MKL_VERBOSE Detected GPU0 Intel(R)_Gen9 Backend:OpenCL VE:72 Stack:1 maxWGsize:256
```

## Call Description Line

### Call Description Line for CPU

In Intel® oneAPI Math Kernel Library Verbose mode, each verbose-enabled function called from your application prints a call description line. The line begins with the `MKL_VERBOSE` character string and uses spaces as delimiters. The format of the rest of the line is subject to change in a future release.

The following table lists information contained in a call description line for Verbose with CPU applications and provides available links for more information:

| Information | Description | Related Links |
|---|---|---|
| The name of the function. | Although the name printed may differ from the name used in the source code of the application (for example, the `cblas_` prefix of CBLAS functions is not printed), you can easily recognize the function by the printed name. | |
| Values of the arguments. | • The values are listed in the order of the formal argument list. The list directly follows the function name, it is parenthesized and comma-separated.<br>• Arrays are printed as addresses (to see the alignment of the data).<br>• Integer scalar parameters passed by reference are printed by value. Zero values are printed for `NULL` references.<br>• Character values are printed without quotes.<br>• For all parameters passed by reference, the values printed are the values *returned by the function*. For example, the printed value of the *`info`* parameter of a LAPACK function is its value after the function execution.<br>• For verbose-enabled functions in the ScaLAPACK domain, in addition to the standard input parameters, information about blocking factors, MPI rank, and process grid is also printed. | |
| Time taken by the function. | • The time is printed in convenient units (seconds, milliseconds, and so on), which are explicitly indicated.<br>• The time may fluctuate from run to run.<br>• The time printed may occasionally be larger than the time actually taken by the function call, especially for small problem sizes and multi-socket machines. | |
| Value of the `MKL_CBWR` environment variable. | The value printed is prefixed with `CNR:` | Getting Started with Conditional Numerical Reproducibility |
| Value of the `MKL_DYNAMIC` environment variable. | The value printed is prefixed with `Dyn:` | `MKL_DYNAMIC` |

| **Information** | **Description** | **Related Links** |
|---|---|---|
| Status of the Intel® oneAPI Math Kernel Librarymemory manager. | The value printed is prefixed with `FastMM:` | Avoiding Memory Leaks in oneMKLfor a description of the Intel® oneAPI Math Kernel Librarymemory manager |
| OpenMP* thread number of the calling thread. | The value printed is prefixed with `TID:` | |
| Values of Intel® oneAPI Math Kernel Library environment variables defining the general and domain-specific numbers of threads, separated by a comma. | The first value printed is prefixed with `NThr:` | oneMKL-specific Environment Variables for Threading Control |

The following is an example of a call description line (with OpenMP threading):

```
MKL_VERBOSE
DGEMM(n,n,1000,1000,240,0x7ffff708bb30,0x7ff2aea4c000,1000,0x7ff28e92b000,240,0x7ffff70
8bb38,0x7ff28e08d000,1000) 1.66ms CNR:OFF Dyn:1 FastMM:1 TID:0 NThr:16
```

The following is an example of a call description line (with TBB threading):

```
MKL_VERBOSE
DGEMM(n,n,1000,1000,240,0x7ffff708bb30,0x7ff2aea4c000,1000,0x7ff28e92b000,240,0x7ffff70
8bb38,0x7ff28e08d000,1000) 1.66ms CNR:OFF Dyn:1 FastMM:1
```

**NOTE** For more information about selected threading, refer to Version Information Line.

The following information is not printed because of limitations of Intel® oneAPI Math Kernel Library Verbose mode:

- Input values of parameters passed by reference if the values were changed by the function.

  For example, if a LAPACK function is called with a workspace query, that is, the value of the *lwork* parameter equals -1 on input, the call description line prints the result of the query and not -1.
- Return values of functions.

  For example, the value returned by the function `ilaenv` is not printed.
- Floating-point scalars passed by reference.

## Call Description Line for GPU

In Intel® oneAPI Math Kernel Library Verbose mode, each verbose-enabled function called from your application prints a call description line. The line begins with the `MKL_VERBOSE` character string and uses spaces as delimiters. The format of the rest of the line may change in a future release.

The following table lists information contained in a call description line for verbose with GPU applications.

**See Also**GPU Information Line

| Information | Description |
|---|---|
| The name of the function | Although the name printed may differ from the name used in the source code of the application, you can easily recognize the function by the printed name. |
| The values of the arguments | • The values are listed in the order of the formal argument list. The list directly follows the function name, and it is parenthesized and comma-separated.<br>• Arrays are printed as addresses (to show the alignment of the data).<br>• Integer scalar parameters passed by reference are printed by value. Zero values are printed for NULL references.<br>• Character values are printed without quotation marks.<br>• For all parameters passed by reference, the values printed are the values returned by the function. |
| Time taken by the function | • If verbose is enabled with timing for GPU applications, kernel executions will become synchronous (previous kernel will block later kernels) and the measured time may include potential data transfers and/or data copies in host and devices.<br>• If Verbose is enabled without timing for GPU applications, time will be printed out as 0.<br>• The time is printed in convenient units (seconds, milliseconds, and so on), which are explicitly indicated.<br>• The time may fluctuate from run to run.<br>• The time printed may occasionally be larger than the time actually taken by the function call, especially for small problem sizes. |
| Device index | The index of the GPU device on which the kernel is being executed will be printed after the character string "GPU" (e.g. GPU0, GPU1, GPU2, etc). Use the index and refer to the GPU information lines for more information about the specific device.<br><br>If the kernel is executed on the host CPU, this field will be empty. |

The following is an example of a call description line:

```
MKL_VERBOSE FFT(dcfi64) 224.30us GPU0
```

**Some Limitations**
For GPU applications, the call description lines may be printed out-of-order (the order of the call description lines printed in the verbose output may not be the order in which the kernels are submitted in the functions) for the following two cases:

- Verbose is enabled without timing and the kernel executions stay asynchronous.
- The kernel is not executed on one of the GPU devices, but on the host CPU (the device index will not be printed in this case).

# Working with the Intel® oneAPI Math Kernel Library Cluster Software

9

Intel® oneAPI Math Kernel Library (oneMKL) includes distributed memory function domains for use on clusters:

- ScaLAPACK
- Cluster Fourier Transform Functions (Cluster FFT)
- Parallel Direct Sparse Solvers for Clusters (Cluster Sparse Solver)

ScaLAPACK, Cluster FFT, and Cluster Sparse Solver are only provided for the Intel® 64 architecture.

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.<br><br>Notice revision #20201201 |

**See Also**

Intel® oneAPI Math Kernel Library Structure

Managing Performance of the Cluster Fourier Transform Functions

## Linking with oneMKL Cluster Software

The Intel® oneAPI Math Kernel Library ScaLAPACK, Cluster FFT, and Cluster Sparse Solver support MPI implementations identified in the*Intel® oneAPI Math Kernel Library Release Notes*.

To link a program that calls ScaLAPACK, Cluster FFT, or Cluster Sparse Solver, you need to know how to link a message-passing interface (MPI) application first.

Use mpi scripts to do this. For example, mpicc or mpif77 are C or FORTRAN 77 scripts, respectively, that use the correct MPI header files. The location of these scripts and the MPI library depends on your MPI implementation. For example, for the default installation of MPICH3, `/opt/mpich/bin/mpicc` and `/opt/mpich/bin/mpif90` are the compiler scripts and `/opt/mpich/lib/libmpi.a` is the MPI library.

Check the documentation that comes with your MPI implementation for implementation-specific details of linking.

To link with ScaLAPACK, Cluster FFT, and/or Cluster Sparse Solver, use the following general form:

```
<MPI linker script> <files to link>                          \
-L <MKL path> [<MKL cluster library>]        \
<BLACS> <MKL core libraries>
```

where the placeholders stand for paths and libraries as explained in the following table:

| | |
|---|---|
| *<MKL cluster library>* | One of libraries for ScaLAPACK or Cluster FFT and appropriate architecture and programming interface (LP64 or ILP64). Available libraries are listed in Appendix C: Directory Structure in Detail. For example, for the LP64 interface, it is `-lmkl_scalapack_lp64` or `-lmkl_cdft_core`. Cluster Sparse Solver does not require an additional computation library. |

| | |
|---|---|
| *<BLACS>* | The BLACS library corresponding to your architecture, programming interface (LP64 or ILP64), and MPI used. Available BLACS libraries are listed in Appendix C: Directory Structure in Detail. Specifically, choose one of `-lmkl_blacs_intelmpi_lp64` or `-lmkl_blacs_intelmpi_ilp64`. |
| *<MKL core libraries>* | Processor optimized kernels, threading library, and system library for threading support, linked as described in Listing Libraries on a Link Line. |
| *<MPI linker script>* | A linker script that corresponds to the MPI version. |

For example, if you are using Intel MPI, want to statically link with ScaLAPACK using the LP64 interface, and have only one MPI process per core (and thus do not use threading), specify the following linker options:

```
-L$MKLPATH -I$MKLINCLUDE $MKLPATH/libmkl_scalapack_lp64.a $MKLPATH/
libmkl_blacs_intelmpi_lp64.a $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_sequential.a
$MKLPATH/libmkl_core.a -static_mpi -lpthread -lm
```

**Tip**
Use the Using the Link-line Advisor to quickly choose the appropriate set of *<MKL cluster Library>*, *<BLACS>*, and *<MKL core libraries>*.

**See Also**
Linking Your Application with the Intel® oneAPI Math Kernel Library
Examples of Linking for Clusters

## Setting the Number of OpenMP* Threads

The OpenMP* run-time library responds to the environment variable `OMP_NUM_THREADS`. Intel® oneAPI Math Kernel Library also has other mechanisms to set the number of OpenMP threads, such as the`MKL_NUM_THREADS` or `MKL_DOMAIN_NUM_THREADS` environment variables (see Using Additional Threading Control).

Make sure that the relevant environment variables have the same and correct values on all the nodes. Intel® oneAPI Math Kernel Library does not set the default number of OpenMP threads to one, but depends on the OpenMP libraries used with the compiler to set the default number. For the threading layer based on the Intel compiler (`libmkl_intel_thread.a`), this value is the number of CPUs according to the OS.

**Caution**
Avoid over-prescribing the number of OpenMP threads, which may occur, for instance, when the number of MPI ranks per node and the number of OpenMP threads per node are both greater than one. The number of MPI ranks per node multiplied by the number of OpenMP threads per node should not exceed the number of hardware threads per node.

If you are using your login environment to set an environment variable, such as `OMP_NUM_THREADS`, remember that changing the value on the head node and then doing your run, as you do on a shared-memory (SMP) system, does not change the variable on all the nodes because `mpirun` starts a fresh default shell on all the nodes. To change the number of OpenMP threads on all the nodes, in `.bashrc`, add a line at the top, as follows:

```
OMP_NUM_THREADS=1; export OMP_NUM_THREADS
```

You can run multiple CPUs per node using MPICH. To do this, build MPICH to enable multiple CPUs per node. Be aware that certain MPICH applications may fail to work perfectly in a threaded environment (see the Known Limitations section in the *Release Notes*. If you encounter problems with MPICH and setting of the number of OpenMP threads is greater than one, first try setting the number of threads to one and see whether the problem persists.

**Important**
For Cluster Sparse Solver, set the number of OpenMP threads to a number greater than one because the implementation of the solver only supports a multithreaded algorithm.

**See Also**
Techniques to Set the Number of Threads

## Using Shared Libraries

All needed shared libraries must be visible on all nodes at run time. To achieve this, set the `DYLD_LIBRARY_PATH` environment variable accordingly.

If Intel® oneAPI Math Kernel Library is installed only on one node, link statically when building your Intel® oneAPI Math Kernel Library applications rather than use shared libraries.

The Intel® compilers or GNU compilers can be used to compile a program that uses Intel® oneAPI Math Kernel Library. However, make sure that the MPI implementation and compiler match up correctly.

## Setting Environment Variables on a Cluster

By default, when you call the MPI launch command `mpiexec`, the entire launching node environment is passed to the MPI processes. However, if there are undefined variables or variables that are different from what is stored in your environment, you can use `-env` or `-genv` options with `mpiexec`. Each of these options take two arguments- the name and the value of the environment variable to be passed.

```
-genv NAME1 VALUE1 -genv NAME2 VALUE2
```

```
-env NAME VALUE -genv
```

See these Intel MPI examples on how to set the value of `OMP_NUM_THREADS` explicitly:

```
mpiexec -genv OMP_NUM_THREADS 2 ....
```

```
mpiexec -n 1 -host first -env OMP_NUM_THREADS 2 test.exe : -n 2 -host second -env
OMP_NUM_THREADS 3 test.exe ....
```

See these Intel MPI examples on how to set the value of `MKL_BLACS_MPI` explicitly:

```
mpiexec -genv MKL_BLACS_MPI INTELMPI ....
```

```
mpiexec -n 1 -host first -env MKL_BLACS_MPI INTELMPI test.exe : -n 1 -host second -env
MKL_BLACS_MPI INTELMPI test.exe.
```

## Interaction with the Message-passing Interface

To improve performance of cluster applications, it is critical for Intel® oneAPI Math Kernel Library to use the optimal number of threads, as well as the correct thread affinity. Usually, the optimal number is the number of available cores per node divided by the number of MPI processes per node. You can set the number of threads using one of the available methods, described inTechniques to Set the Number of Threads.

If the number of threads is not set, Intel® oneAPI Math Kernel Library checks whether it runs under MPI provided by the Intel® MPI Library. If this is true, the following environment variables define Intel® oneAPI Math Kernel Library threading behavior:

- `I_MPI_THREAD_LEVEL`
- `MKL_MPI_PPN`
- `I_MPI_NUMBER_OF_MPI_PROCESSES_PER_NODE`
- `I_MPI_PIN_MAPPING`
- `OMPI_COMM_WORLD_LOCAL_SIZE`
- `MPI_LOCALNRANKS`

The threading behavior depends on the value of `I_MPI_THREAD_LEVEL` as follows:

- 0 or undefined.

  Intel® oneAPI Math Kernel Library considers that thread support level of Intel MPI Library is`MPI_THREAD_SINGLE` and defaults to sequential execution.
- 1, 2, or 3.

  This value determines Intel® oneAPI Math Kernel Library conclusion of the thread support level:

  - 1 - `MPI_THREAD_FUNNELED`
  - 2 - `MPI_THREAD_SERIALIZED`
  - 3 - `MPI_THREAD_MULTIPLE`

  In all these cases, Intel® oneAPI Math Kernel Library determines the number of MPI processes per node using the other environment variables listed and defaults to the number of threads equal to the number of available cores per node divided by the number of MPI processes per node.

**Important**
Instead of relying on the discussed implicit settings, explicitly set the number of threads for Intel® oneAPI Math Kernel Library.

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201 |

### See Also

Intel® Software Documentation Library for more information on Intel MPI Library
for more information on Intel MPI Library

## Using a Custom Message-Passing Interface

While different message-passing interface (MPI) libraries are compatible at the application programming interface (API) level, they are often incompatible at the application binary interface (ABI) level. Therefore, Intel® oneAPI Math Kernel Library provides a set of prebuilt BLACS libraries that support certain MPI libraries, but this, however, does not enable use of Intel® oneAPI Math Kernel Library with other MPI libraries. To fill this gap, Intel® oneAPI Math Kernel Library also includes the MKL MPI wrapper, which provides an MPI-independent ABI to Intel® oneAPI Math Kernel Library. The adaptor is provided as source code. To use Intel® oneAPI Math Kernel Library with an MPI library that is not supported by default, you can use the adapter to build custom static or dynamic BLACS libraries and use them similarly to the prebuilt libraries.

### Building a Custom BLACS Library

The MKL MPI wrapper is located in the `<mkl directory>/interfaces/mklmpi` directory.

To build a custom BLACS library, from the above directory run the `make` command.

For example: the command

```
make libintel64
```

builds a static custom BLACS library `libmkl_blacs_custom_lp64.a` using the MPI compiler from the current shell environment. Look into the `<mkl directory>/interfaces/mklmpi/makefile` for targets and variables that define how to build the custom library. In particular, you can specify the compiler through the `MPICC` variable.

For more control over the building process, refer to the documentation available through the command

```
make help
```

### Using a Custom BLACS Library

Use custom BLACS libraries exactly the same way as you use the prebuilt BLACS libraries, but pass the custom library to the linker. For example, instead of passing the `libmkl_blacs_mpich_lp64.a` library, pass `libmkl_blacs_custom_lp64.a`.

#### See Also

Linking with Intel® oneAPI Math Kernel Library Cluster Software

## Examples of Linking for Clusters

This section provides examples of linking with ScaLAPACK, Cluster FFT, and Cluster Sparse Solver.

Note that a binary linked with the Intel® oneAPI Math Kernel Library cluster function domains runs the same way as any other MPI application (refer to the documentation that comes with your MPI implementation).For instance, the script `mpirun` is used in the case of MPICH, and the number of MPI processes is set by `-np`. In the case of MPICH, start the daemon before running your application; the execution is driven by the script `mpiexec`.

For further linking examples, see the support website for Intel products at https://software.intel.com/content/www/us/en/develop/support.html.

#### See Also

Directory Structure in Detail

### Examples for Linking a C Application

- Main module is in C.
- You are using the Intel® oneAPI DPC++/C++ CompilerIntel® C++ Compiler.
- You are using MPICH.
- Intel® oneAPI Math Kernel Library functions use LP64 interfaces.
- The `PATH` environment variable contains a directory with the MPI linker scripts.
- `$MKLPATH` is a user-defined variable containing `<mkl_directory>/lib`.

To link dynamically with ScaLAPACK for a cluster of systems based on the Intel® 64 architecture, use the following link line:

```
mpicc <user files to link>                  \
   -L$MKLPATH                               \
   -lmkl_scalapack_lp64                     \
   -lmkl_blacs_mpich_lp64                   \
   -lmkl_intel_lp64                         \
   -lmkl_intel_thread -lmkl_core            \
   -liomp5 -lpthread
```

To link statically with Cluster FFT for a cluster of systems based on the Intel® 64 architecture, use the following link line:

```
mpicc <user files to link>             \
   $MKLPATH/libmkl_cdft_core.a            \
   $MKLPATH/libmkl_blacs_mpich_lp64.a     \
   $MKLPATH/libmkl_intel_lp64.a           \
   $MKLPATH/libmkl_intel_thread.a         \
   $MKLPATH/libmkl_core.a                 \
   -liomp5 -lpthread
```

To link dynamically with Cluster Sparse Solver for a cluster of systems based on the Intel® 64 architecture, use the following link line:

```
mpicc <user files to link>                       \
   -L$MKLPATH                                    \
   -lmkl_blacs_mpich_lp64                        \
   -lmkl_intel_lp64                              \
   -lmkl_intel_thread -lmkl_core                 \
   -liomp5 -lpthread
```

**See Also**
Linking with oneMKL Cluster Software
Using the Link-line Advisor

## Examples for Linking a Fortran Application

These examples illustrate linking of an application under the following conditions:

- Main module is in Fortran.
- You are using the Intel® Fortran Compiler.
- You are using the MPICH library.
- Intel® oneAPI Math Kernel Library functions use LP64 interfaces.
- The `PATH` environment variable contains a directory with the MPI linker scripts.
- `$MKLPATH` is a user-defined variable containing `<mkl_directory>/lib`.

To link dynamically with ScaLAPACK for a cluster of systems based on the Intel® 64 architecture, use the following link line:

```
mpiifort <user files to link>                         \
   -L$MKLPATH                                         \
   -lmkl_scalapack_lp64                               \
   -lmkl_blacs_mpich_lp64                             \
   -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core     \
   -liomp5 -lpthread
```

To link statically with Cluster FFT for a cluster of systems based on the Intel® 64 architecture, use the following link line:

```
mpiifort <user files to link>                         \
   $MKLPATH/libmkl_cdft_core.a                        \
   $MKLPATH/libmkl_blacs_mpich_lp64.a                 \
   $MKLPATH/libmkl_intel_lp64.a                       \
   $MKLPATH/libmkl_intel_thread.a                     \
   $MKLPATH/libmkl_core.a                             \
   -liomp5 -lpthread
```

To link statically with Cluster Sparse Solver for a cluster of systems based on the Intel® 64 architecture, use the following link line:

```
mpiifort <user files to link>                         \
   $MKLPATH/libmkl_blacs_mpich_lp64.a                 \
   $MKLPATH/libmkl_intel_lp64.a                       \
   $MKLPATH/libmkl_intel_thread.a                     \
   $MKLPATH/libmkl_core.a                             \
   -liomp5 -lpthread
```

**See Also**
Linking with oneMKL Cluster Software
Using the Link-line Advisor

# *Managing Behavior of the Intel® oneAPI Math Kernel Library with Environment Variables* 10

**See Also**

Intel® oneAPI Math Kernel Library-specific Environment Variables for Threading Control

Specifying the Code Branches
  for how to use an environment variable to specify the code branch for Conditional Numerical Reproducibility
Using Intel® oneAPI Math Kernel Library Verbose Mode
  for how to use an environment variable to set the verbose mode

## Managing Behavior of Function Domains with Environment Variables

### Setting the Default Mode of Vector Math with an Environment Variable

Intel® oneAPI Math Kernel Library (oneMKL) enables overriding the default setting of the Vector Mathematics (VM) global mode using the`MKL_VML_MODE` environment variable.

Because the mode is set or can be changed in different ways, their precedence determines the actual mode used. The settings and function calls that set or change the VM mode are listed below, with the precedence growing from lowest to highest:

1. The default setting
2. The `MKL_VML_MODE` environment variable
3. A call `vmlSetMode` function
4. A call to any VM function other than a service function

For more details, see the Vector Mathematical Functions section in the *Intel® oneAPI Math Kernel Library Developer Reference* and the description of the `vmlSetMode` function in particular.

To set the `MKL_VML_MODE` environment variable, use the following command in your command shell:

- For the bash shell:

  ```
  export MKL_VML_MODE=<mode-string>
  ```
- For a C shell (csh or tcsh):

  ```
  setenv MKL_VML_MODE <mode-string>
  ```

In these commands, *`<mode-string>`* controls error handling behavior and computation accuracy, consists of one or several comma-separated values of the *`mode`* parameter listed in the table below, and meets these requirements:

- Not more than one accuracy control value is permitted
- Any combination of error control values except `VML_ERRMODE_DEFAULT` is permitted
- No denormalized numbers control values are permitted

**Values of the *mode* Parameter**

| Value of *mode* | Description |
|---|---|
| Accuracy Control | |
| `VML_HA` | high accuracy versions of VM functions |

| Value of *mode* | Description |
|---|---|
| `VML_LA` | low accuracy versions of VM functions |
| `VML_EP` | enhanced performance accuracy versions of VM functions |
| Denormalized Numbers Handling Control | |
| `VML_FTZDAZ_ON` | Faster processing of denormalized inputs is enabled. |
| `VML_FTZDAZ_OFF` | Faster processing of denormalized inputs is disabled. |
| `VML_FTZDAZ_CURRENT` | Keep the current CPU settings for denormalized inputs. |
| Error Mode Control | |
| `VML_ERRMODE_IGNORE` | On computation error, VM Error status is updated, but otherwise no action is set. Cannot be combined with other `VML_ERRMODE` settings. |
| `VML_ERRMODE_NOERR` | On computation error, VM Error status is not updated and no action is set. Cannot be combined with other `VML_ERRMODE` settings. |
| `VML_ERRMODE_STDERR` | On error, the error text information is written to *stderr*. |
| `VML_ERRMODE_EXCEPT` | On error, an exception is raised. |
| `VML_ERRMODE_CALLBACK` | On error, an additional error handler function is called. |
| `VML_ERRMODE_DEFAULT` | On error, an exception is raised and an additional error handler function is called. |

These commands provide an example of valid settings for the `MKL_VML_MODE` environment variable in your command shell:

- For the bash shell:

```
export MKL_VML_MODE=VML_LA,VML_ERRMODE_ERRNO,VML_ERRMODE_STDERR
```

- For a C shell (csh or tcsh):

```
setenv MKL_VML_MODE VML_LA,VML_ERRMODE_ERRNO,VML_ERRMODE_STDERR
```

**NOTE**
VM ignores the `MKL_VML_MODE` environment variable in the case of incorrect or misspelled settings of *`mode`*.

## Managing Performance of the Cluster Fourier Transform Functions

Performance of Intel® oneAPI Math Kernel Library Cluster FFT (CFFT) in different applications mainly depends on the cluster configuration, performance of message-passing interface (MPI) communications, and configuration of the run. Note that MPI communications usually take approximately 70% of the overall CFFT compute time.For more flexibility of control over time-consuming aspects of CFFT algorithms, Intel® oneAPI Math Kernel Library provides the`MKL_CDFT` environment variable to set special values that affect CFFT performance. To improve performance of your application that intensively calls CFFT, you can use the environment variable to set optimal values for you cluster, application, MPI, and so on.

The `MKL_CDFT` environment variable has the following syntax, explained in the table below:

```
MKL_CDFT=option1[=value1],option2[=value2],…,optionN[=valueN]
```

**Important**
While this table explains the settings that usually improve performance under certain conditions, the actual performance highly depends on the configuration of your cluster. Therefore, experiment with the listed values to speed up your computations.

| Option | Possible Values | Description |
|---|---|---|
| `alltoallv` | 0 (default) | Configures CFFT to use the standard `MPI_Alltoallv` function to perform global transpositions. |
| | 1 | Configures CFFT to use a series of calls to `MPI_Isend` and `MPI_Irecv` instead of the `MPI_Alltoallv` function. |
| | 4 | Configures CFFT to merge global transposition with data movements in the local memory. CFFT performs global transpositions by calling `MPI_Isend` and `MPI_Irecv` in this case.<br><br>Use this value in a hybrid case (MPI + OpenMP), especially when the number of processes per node equals one. |
| `wo_omatcopy` | 0 | Configures CFFT to perform local FFT and local transpositions separately.<br><br>CFFT usually performs faster with this value than with `wo_omatcopy` = 1 if the configuration parameter `DFTI_TRANSPOSE` has the value of `DFTI_ALLOW`. See the *Intel® oneAPI Math Kernel Library Developer Reference* for details. |
| | 1 | Configures CFFT to merge local FFT calls with local transpositions.<br><br>CFFT usually performs faster with this value than with `wo_omatcopy` = 0 if `DFTI_TRANSPOSE` has the value of `DFTI_NONE`. |
| | -1 (default) | Enables CFFT to decide which of the two above values to use depending on the value of `DFTI_TRANSPOSE`. |
| `enable_soi` | Not applicable | A flag that enables low-communication Segment Of Interest FFT (SOI FFT) algorithm for one-dimensional complex-to-complex CFFT, which requires fewer MPI communications than the standard nine-step (or six-step) algorithm.<br><br>**Caution**<br>While using fewer MPI communications, the SOI FFT algorithm incurs a minor loss of precision (about one decimal digit). |

The following example illustrates usage of the environment variable:

```
export MKL_CDFT=wo_omatcopy=1,alltoallv=4,enable_soi
mpirun -ppn 2 -n 16 ./mkl_cdft_app
```

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201

## Managing Invalid Input Checking in LAPACKE Functions

The high-level interface includes an optional, on by default, NaN check on all matrix inputs before calling any LAPACK routine. This option affects all routines. If an input matrix contains any NaNs, the input parameter corresponding to this matrix is flagged with a return value error. For example, if the fifth parameter is found to contain a NaN, the routine returns the value, `-5`. The middle-level interface does not contain the NaN check.

NaN checking on matrix input can be expensive. By default, NaN checking is turned **on**. LAPACKE provides a way to set it through the environment variable:

- Setting environment variable `LAPACKE_NANCHECK` to `0` turns OFF NaN-checking
- Setting environment variable `LAPACKE_NANCHECK` to `1` turns ON NaN-checking

The other way is the call the `LAPACKE_set_nancheck` function; see the Developer Reference for C's LAPACK Auxiliary Routines section for more information.

Note that the NaN-checking flag value set by the call to `LAPACKE_set_nancheck` always has higher priority than the environment variable, `LAPACKE_NANCHECK`.

## Instruction Set Specific Dispatching on Intel® Architectures

Intel® oneAPI Math Kernel Library automatically queries and then dispatches the code path supported on your Intel® processor to the optimal instruction set architecture (ISA) by default. The `MKL_ENABLE_INSTRUCTIONS` environment variable or the `mkl_enable_instructions` support function enables you to dispatch to an ISA-specific code path of your choice. For example, you can run the Intel® Advanced Vector Extensions (Intel® AVX) code path on an Intel processor based on Intel® Advanced Vector Extensions 2 (Intel® AVX2). This feature is not available on non-Intel processors.

In some cases Intel® oneAPI Math Kernel Library also provides support for upcoming architectures ahead of hardware availability, but the library does not automatically dispatch the code path specific to an upcoming ISA by default. If for your exploratory work you need to enable an ISA for an Intel processor that is not yet released or if you are working in a simulated environment, you can use the`MKL_ENABLE_INSTRUCTIONS` environment variable or `mkl_enable_instructions` support function.

The following table lists possible values of `MKL_ENABLE_INSTRUCTIONS` alongside the corresponding ISA supported by a given processor. `MKL_ENABLE_INSTRUCTIONS`dispatches to the default ISA if the ISA requested is not supported on the particular Intel processor. For example, if you request to run the Intel AVX512 code path on a processor based on Intel AVX2, Intel® oneAPI Math Kernel Library runs the Intel AVX2 code path. The table also explains whether the ISA is dispatched by default on the processor that supports this ISA.

| Value of **`MKL_ENABLE_INSTRUCTIONS`** | ISA | Dispatched by Default |
|---|---|---|
| `AVX512` | Intel® Advanced Vector Extensions (Intel® AVX-512) for systems based on Intel® Xeon® processors | Yes |
| `AVX512_E1` | Intel® Advanced Vector Extensions (Intel® AVX-512) with support for Vector Neural Network Instructions. | Yes |
| `AVX512_E2` | ICX: Intel® Advanced Vector Extensions (Intel® AVX-512) enabled processors. | Yes |
| `AVX512_E3` | Intel® Advanced Vector Extensions 512 (Intel® AVX-512) with support of Vector Neural Network Instructions supporting BF16 enabled processors. | Yes |
| `AVX512_E4` | Intel® Advanced Vector Extensions 512 (Intel® AVX-512) with Intel® Deep Learning Boost (Intel® DL Boost) and bfloat16 support and Intel® Advanced Matrix Extensions (Intel® AMX) with bfloat16 and 8-bit integer support. | Yes |
| `AVX2` | Intel® AVX2 | Yes |

| Value of `MKL_ENABLE_INSTRUCTIONS` | ISA | Dispatched by Default |
|---|---|---|
| `AVX2_E1` | Intel® Advanced Vector Extensions 2 (Intel® AVX2) with support for Intel® Deep Learning Boost (Intel® DL Boost). | Yes |
| `AVX` | Intel® AVX | Yes |

For more details about the `mkl_enable_instructions` function, including the argument values, see the *Intel® oneAPI Math Kernel Library Developer Reference*.

For example:

- To configure the library not to dispatch more recent architectures than Intel AVX2, do one of the following:
  - Call

    ```
    mkl_enable_instructions(MKL_ENABLE_AVX2)
    ```
  - Set the environment variable:
    - For the bash shell:

      ```
      export MKL_ENABLE_INSTRUCTIONS=AVX2
      ```
    - For a C shell (csh or tcsh):

      ```
      setenv MKL_ENABLE_INSTRUCTIONS AVX2
      ```

**NOTE**
Settings specified by the `mkl_enable_instructions` function take precedence over the settings specified by the `MKL_ENABLE_INSTRUCTIONS` environment variable.

**Product and Performance Information**

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Notice revision #20201201

# *Configuring Your Integrated Development Environment to Link with Intel® oneAPI Math Kernel Library* 11

## Configuring the Apple Xcode* Developer Software to Link with Intel® Math Kernel Library

This section provides information on configuring the Apple Xcode* developer software for linking with Intel® oneAPI Math Kernel Library.

To configure your Xcode developer software to link with Intel® oneAPI Math Kernel Library, you need to perform the steps explained below. The specific instructions for performing these steps depend on your version of the Xcode developer software. Please refer to the Xcode Help for more details.

To configure your Xcode developer software, do the following:

1. Open your project that uses Intel® oneAPI Math Kernel Library and select the target you are going to build.
2. Add the Intel® oneAPI Math Kernel Library include path, that is,`<mkl directory>/include`, to the header search paths.
3. Add the Intel® oneAPI Math Kernel Library library path for the target architecture to the library search paths. For example, for the Intel® 64 architecture, add`<mkl directory>/lib/intel64`.
4. Specify the linker options for the Intel® oneAPI Math Kernel Library and system libraries to link with your application. For example, you may need to specify:`-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -lpthread -lm`.
5. (Optional, needed only for dynamic linking) For the active executable, add the environment variable `DYLD_LIBRARY_PATH` with the value of `<mkl directory>/lib`.

**See Also**

Notational Conventions

Linking in Detail

# Intel® oneAPI Math Kernel Library Benchmarks

12

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. <br> Notice revision #20201201 |

## Intel Optimized LINPACK Benchmark for macOS*

Intel Optimized LINPACK Benchmark for macOS* is a generalization of the LINPACK 1000 benchmark. It solves a dense (`real`*8) system of linear equations (*Ax=b*), measures the amount of time it takes to factor and solve the system, converts that time into a performance rate, and tests the results for accuracy. The generalization is in the number of equations (*N*) it can solve, which is not limited to 1000. It uses partial pivoting to assure the accuracy of the results.

Do not use this benchmark to report LINPACK 100 performance because that is a compiled-code only benchmark. This is a shared-memory (SMP) implementation which runs on a single platform. Do not confuse this benchmark with LINPACK, the library, which has been expanded upon by the LAPACK library.

Intel provides optimized versions of the LINPACK benchmarks to help you obtain high LINPACK benchmark results on your genuine Intel processor systems more easily than with the High Performance Linpack (HPL) benchmark.

Additional information on this software, as well as on other Intel® software performance products, is available at https://software.intel.com/content/www/us/en/develop/tools.html.

### Acknowledgement

This product includes software developed at the University of Tennessee, Knoxville, Innovative Computing Laboratories.

### Contents of the Intel® Optimized LINPACK Benchmark

The Intel Optimized LINPACK Benchmark for macOS* contains the following files, located in the `./benchmarks/linpack/`subdirectory of the Intel® oneAPI Math Kernel Library (oneMKL) directory:

| **File in** `./benchmarks/linpack/` | **Description** |
|---|---|
| `linpack_cd64` | The 64-bit program executable for a system using Intel® Core™ microarchitecture on macOS. |
| `runme64` | A sample shell script for executing a pre-determined problem set for `linpack_cd64`. |
| `help.lpk` | Simple help file. |
| `xhelp.lpk` | Extended help file. |
| These files are not available immediately after installation and appear as a result of execution of an appropriate `runme` script. | |
| `lin_cd64.txt` | Result of the `runme64` script execution. |

### See Also

High-level Directory Structure

## Running the Software

To obtain results for the pre-determined sample problem sizes on a given system, type:

```
./runme64
```

To run the software for other problem sizes, see the extended help included with the program. You can view extended help by running the program executable with the `-e` option:

```
./linpack_cd64 -e
```

The pre-defined data input file `lininput` isan example. Different systems have different amounts of memory and therefore require new input files. The extended help can give insight into proper ways to change the sample input files.

`lininput` requires at least 2 GB of memory.

If the system has less memory than the above sample data input requires, you may need to edit or create your own data input files, as explained in the extended help.

The Intel Optimized LINPACK Benchmark determines the optimal number of OpenMP threads to use. To run a different number, you can set the `OMP_NUM_THREADS` or `MKL_NUM_THREADS` environment variable inside a sample script. If you run the Intel Optimized LINPACK Benchmark without setting the number of threads, it defaults to the number of physical cores.

| **Product and Performance Information** |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.<br><br>Notice revision #20201201 |

## Known Limitations of the Intel® Optimized LINPACK Benchmark

The following limitations are known for the Intel Optimized LINPACK Benchmark for macOS*:

- Intel Optimized LINPACK Benchmark supports only OpenMP threading
- If an incomplete data input file is given, the binaries may either hang or fault. See the sample data input files and/or the extended help for insight into creating a correct data input file.
- The binary will hang if it is not given an input file or any other arguments.

# Intel® oneAPI Math Kernel Library Language Interfaces Support A

**See Also**
Mixed-language Programming with Intel® oneAPI Math Kernel Library

## Language Interfaces Support, by Function Domain

The following table shows language interfaces that Intel® oneAPI Math Kernel Library (oneMKL) provides for each function domain. However, Intel® oneAPI Math Kernel Library routines can be called from other languages using mixed-language programming. SeeMixed-language Programming with the Intel Math Kernel Library for an example of how to call Fortran routines from C/C++.

| Function Domain | Fortran interface | C/C++ interface |
|---|---|---|
| Basic Linear Algebra Subprograms (BLAS) | Yes | through CBLAS |
| BLAS-like extension transposition routines | Yes | Yes |
| Sparse BLAS Level 1 | Yes | through CBLAS |
| Sparse BLAS Level 2 and 3 | Yes | Yes |
| LAPACK routines for solving systems of linear equations | Yes | Yes |
| LAPACK routines for solving least-squares problems, eigenvalue and singular value problems, and Sylvester's equations | Yes | Yes |
| Auxiliary and utility LAPACK routines | Yes | Yes |
| Parallel Basic Linear Algebra Subprograms (PBLAS) | Yes | |
| ScaLAPACK | Yes | † |
| Direct Sparse Solvers/ Intel® oneAPI Math Kernel Library PARDISO, a direct sparse solver based on Parallel Direct Sparse Solver (PARDISO*) | Yes | Yes |
| Parallel Direct Sparse Solvers for Clusters | Yes | Yes |
| Other Direct and Iterative Sparse Solver routines | Yes | Yes |
| Vector Mathematics (VM) | Yes | Yes |
| Vector Statistics (VS) | Yes | Yes |
| Fast Fourier Transforms (FFT) | Yes | Yes |
| Cluster FFT | Yes | Yes |
| Trigonometric Transforms | Yes | Yes |
| Fast Poisson, Laplace, and Helmholtz Solver (Poisson Library) | Yes | Yes |
| Optimization (Trust-Region) Solver | Yes | Yes |

| Function Domain | Fortran int erface | C/C++ interface |
|---|---|---|
| Data Fitting | Yes | Yes |
| Extended Eigensolver | Yes | Yes |
| Support functions (including memory allocation) | Yes | Yes |

[†] Supported using a mixed language programming call. See Include Files for the respective header file.

## Include Files

The table below lists Intel® oneAPI Math Kernel Library include files.

| Function Domain/ Purpose | Fortran Include Files | C/C++ Include Files |
|---|---|---|
| All function domains | `mkl.fi` | `mkl.h` |
| BLACS | | `mkl_blacs.h`[‡‡] |
| BLAS | `blas.f90`<br>`mkl_blas.fi`[†] | `mkl_blas.h`[‡] |
| BLAS-like Extension Transposition Routines | `mkl_trans.fi`[†] | `mkl_trans.h`[‡] |
| CBLAS Interface to BLAS | | `mkl_cblas.h`[‡] |
| Sparse BLAS | `mkl_spblas.fi`[†] | `mkl_spblas.h`[‡] |
| LAPACK | `lapack.f90`<br>`mkl_lapack.fi`[†] | `mkl_lapack.h`[‡] |
| C Interface to LAPACK | | `mkl_lapacke.h`[‡] |
| PBLAS | | `mkl_pblas.h`[‡‡] |
| ScaLAPACK | | `mkl_scalapack.h`[‡‡] |
| Intel® oneAPI Math Kernel Library PARDISO | `mkl_pardiso.f90`<br>`mkl_pardiso.fi`[†] | `mkl_pardiso.h`[‡] |
| Parallel Direct Sparse Solvers for Clusters | `mkl_cluster_ sparse_solver.f90` | `mkl_cluster_ sparse_solver.h`[‡] |
| Direct Sparse Solver (DSS) | `mkl_dss.f90`<br>`mkl_dss.fi`[†] | `mkl_dss.h`[‡] |
| RCI Iterative Solvers | | |
| ILU Factorization | `mkl_rci.f90`<br>`mkl_rci.fi`[†] | `mkl_rci.h`[‡] |
| Optimization Solver | `mkl_rci.f90`<br>`mkl_rci.fi`[†] | `mkl_rci.h`[‡] |
| Vector Mathematics | `mkl_vml.90`<br>`mkl_vml.fi`[†] | `mkl_vml.h`[‡] |
| Vector Statistics | `mkl_vsl.f90`<br>`mkl_vsl.fi`[†] | `mkl_vsl.h`[‡] |

| **Function Domain/ Purpose** | **Fortran Include Files** | **C/C++ Include Files** |
|---|---|---|
| Fast Fourier Transforms | `mkl_dfti.f90` | `mkl_dfti.h`‡ |
| Cluster Fast Fourier Transforms | `mkl_cdft.f90` | `mkl_cdft.h`‡‡ |
| Partial Differential Equations Support | | |
| Trigonometric Transforms | `mkl_trig_transforms.f90` | `mkl_trig_transform.h`‡ |
| Poisson Solvers | `mkl_poisson.f90` | `mkl_poisson.h`‡ |
| Data Fitting | `mkl_df.f90` | `mkl_df.h`‡ |
| Extended Eigensolver | `mkl_solvers_ee.fi`† | `mkl_solvers_ee.h`‡ |
| Support functions | `mkl_service.f90` `mkl_service.fi`† | `mkl_service.h`‡ |
| Declarations for replacing memory allocation functions. See Redefining Memory Functions for details. | | `i_malloc.h` |
| Auxiliary macros to determine the version of Intel® oneAPI Math Kernel Library at compile time. | `mkl_version` | `mkl_version`‡ |

† You can use the `mkl.fi` include file in your code instead.

‡ You can include the `mkl.h` header file in your code instead.

‡‡ Also include the `mkl.h` header file in your code.

**See Also**

Language Interfaces Support, by Function Domain

# Support for Third-Party Interfaces B

## FFTW Interface Support

Intel® oneAPI Math Kernel Library (oneMKL) offers two collections of wrappers for the FFTW interface (www.fftw.org). The wrappers are the superstructure of FFTW to be used for calling the Intel® oneAPI Math Kernel Library Fourier transform functions. These collections correspond to the FFTW versions 2.x and 3.x and the Intel® oneAPI Math Kernel Library versions 7.0 and later.

These wrappers enable using Intel® oneAPI Math Kernel Library Fourier transforms to improve the performance of programs that use FFTW without changing the program source code. See the "*FFTW Interface to Intel® oneAPI Math Kernel Library*" appendix in the *Intel® oneAPI Math Kernel Library Developer Reference* for details on the use of the wrappers.

**Important**
For ease of use, the FFTW3 interface is also integrated in Intel® oneAPI Math Kernel Library.

**Caution**
The FFTW2 and FFTW3 interfaces are not compatible with each other. Avoid linking to both of them. If you must do so, first modify the wrapper source code for FFTW2:

1. Change every instance of `fftw_destroy_plan` in the `fftw2xc` interface to `fftw2_destroy_plan`.
2. Change all the corresponding file names accordingly.
3. Rebuild the pertinent libraries.

# Directory Structure in Detail

C

Tables in this section show contents of the `<mkl directory>/lib` directory.

| Product and Performance Information |
|---|
| Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.<br>Notice revision #20201201 |

**See Also**
High-level Directory Structure
Using Language-Specific Interfaces with Intel® oneAPI Math Kernel Library

Intel® oneAPI Math Kernel Library Benchmarks

## Static Libraries in the `lib` directory

Some of the libraries in this directory are optional. However, some optional libraries are installed by default, while the rest are not. To get those libraries that are not installed by default, explicitly select the specified optional component during installation.

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| **Interface Layer** | | | |
| `libmkl_intel_lp64.a` | Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support LP64 interface | | |
| `libmkl_intel_ilp64.a` | Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support ILP64 interface | | |
| `libmkl_blas95_lp64.a` | Fortran 95 interface library for BLAS for the Intel® Fortran compiler. To be used on Intel® 64 architecture systems to support LP64 interface. | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| `libmkl_blas95_ilp64.a` | Fortran 95 interface library for BLAS for the Intel® Fortran compiler. | Fortran 95 interfaces for BLAS and LAPACK | Yes |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| | To be used on Intel® 64 architecture systems to support ILP64 interface. | | |
| `libmkl_lapack95_lp64.a` | Fortran 95 interface library for LAPACK for the Intel® Fortran compiler. To be used on Intel® 64 architecture systems to support LP64 interface. | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| `libmkl_lapack95_ilp64.a` | Fortran 95 interface library for LAPACK for the Intel® Fortran compiler. To be used on Intel® 64 architecture systems to support ILP64 interface. | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| **Threading Layer** | | | |
| `libmkl_intel_thread.a` | OpenMP threading library for the Intel compilers | | |
| `libmkl_tbb_thread.a` | Intel® Threading Building Blocks (Intel® TBB) threading library for the Intel compilers | Intel TBB threading support | Yes |
| `libmkl_sequential.a` | Sequential library | | |
| **Computational Layer** | | | |
| `libmkl_core.a` | Kernel library | | |
| **Cluster Libraries** | | | |
| `libmkl_scalapack_lp64.a` | ScaLAPACK routine library supporting the LP64 interface | Cluster support | |
| `libmkl_scalapack_ilp64.a` | ScaLAPACK routine library supporting the ILP64 interface | Cluster support | |
| `libmkl_cdft_core.a` | Cluster version of FFT functions | Cluster support | |
| `libmkl_blacs_mpich_lp64.a` | LP64 version of BLACS routines for MPICH | Cluster support | |
| `libmkl_blacs_mpich_ilp64.a` | ILP64 version of BLACS routines for MPICH | Cluster support | |

## Dynamic Libraries in the `lib` directory

Some of the libraries in this directory are optional. However, some optional libraries are installed by default, while the rest are not. To get those libraries that are not installed by default, explicitly select the specified optional component during installation.

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| `libmkl_rt.dylib` | Single Dynamic Library | | |
| **Interface Layer** | | | |
| `libmkl_intel_lp64.dylib` | Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support LP64 interface | | |
| `libmkl_intel_ilp64.dylib` | Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support ILP64 interface | | |
| **Threading Layer** | | | |
| `libmkl_intel_thread.dylib` | OpenMP threading library for the Intel compilers | | |
| `libmkl_tbb_thread.dylib` | Intel TBB threading library for the Intel compilers | Intel TBB threading support | Yes |
| `libmkl_sequential.dylib` | Sequential library | | |
| **Computational Layer** | | | |
| `libmkl_core.dylib` | Contains the dispatcher for dynamic load of the processor-specific kernel library | | |
| `libmkl_lapack.dylib` | Routines and drivers for LAPACK, DSS, and Intel® oneAPI Math Kernel Library PARDISO | | |
| `libmkl_mc.dylib` | 64-bit kernel for processors based on the Intel® Core™ microarchitecture | | |
| `libmkl_mc3.dylib` | 64-bit kernel for the Intel® Core™ i7 processors | | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| `libmkl_avx.dylib` | Kernel library for Intel® Advanced Vector Extensions (Intel® AVX) | | |
| `libmkl_avx2.dylib` | Kernel library for Intel® Advanced Vector Extensions 2 (Intel® AVX2) | | |
| `libmkl_vml_mc.dylib` | 64-bit Vector Mathematics (VM)/ Vector Statistics (VS)/ Data Fitting (DF) for processors based on the Intel® Core™ microarchitecture | | |
| `libmkl_vml_mc2.dylib` | 64-bit VM/VS/DF for 45nm Hi-k Intel® Core™2 and the Intel Xeon® processor families | | |
| `libmkl_vml_mc3.dylib` | 64-bit VM/VS/DF for the Intel® Core™ i7 processors | | |
| `libmkl_vml_avx.dylib` | VM/VS/DF optimized for the Intel® Advanced Vector Extensions (Intel® AVX) | | |
| `libmkl_vml_avx2.dylib` | VM/VS/DF optimized for Intel® Advanced Vector Extensions 2 (Intel® AVX2) | | |
| `libmkl_vml_cmpt.dylib` | VM/VS/DF library for conditional numerical reproducibility | | |
| **Cluster Libraries** | | | |
| `libmkl_scalapack_lp64.dylib` | ScaLAPACK routine library supporting the LP64 interface | Cluster support | |
| `libmkl_scalapack_ilp64.dylib` | ScaLAPACK routine library supporting the ILP64 interface | Cluster support | |
| `libmkl_cdft_core.dylib` | Cluster version of FFT functions | Cluster support | |
| `libmkl_blacs_mpich_lp64.dylib` | LP64 version of BLACS routines for MPICH | Cluster support | |
| `libmkl_blacs_mpich_ilp64.dylib` | ILP64 version of BLACS routines for MPICH | Cluster support | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | **Name** | **Installed by Default** |
| **Message Catalogs** | | | |
| `locale/en_US/mkl_msg.cat` | Catalog of Intel® oneAPI Math Kernel Library (oneMKL) messages in English | | |

# *Index*