# Intel® Math Kernel Library for Windows* OS

## Developer Guide

*Intel® MKL 11.3 - Windows* OS*

Revision: 049

What's New

Legal Information

# Contents

## Chapter 11: Managing Behavior of the Intel(R) Math Kernel Library with Environment Variables



## Chapter 12: Programming with Intel® Math Kernel Library in Integrated Development Environments (IDE)



## Chapter 13: LINPACK and MP LINPACK Benchmarks



## Appendix A: Intel® Math Kernel Library Language Interfaces Support



## Appendix B: Support for Third-Party Interfaces

# *Legal Information*



| **Optimization Notice** |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

# 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 MKL support website at http://www.intel.com/software/products/support/.

You can get context-sensitive help when editing your code in the Microsoft Visual Studio* integrated development environment (IDE). See Getting Assistance for Programming in the Microsoft Visual Studio* IDE for details.

# *Introducing the Intel® Math Kernel Library*

Intel® Math Kernel Library (Intel® MKL) 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. Intel MKL 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.

Intel MKL 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).
- Intel MKL 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 Intel MKL 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® MKL Reference Manual*.

Intel MKL is optimized for the latest Intel processors, including processors with multiple cores (see the *Intel MKL Release Notes* for the full list of supported processors). Intel MKL also performs well on non-Intel processors.

For Windows* and Linux* systems based on Intel® 64 Architecture, Intel MKL also includes support for the Intel® Many Integrated Core (Intel® MIC) Architecture and provides libraries to help you port your applications to Intel MIC Architecture.

**NOTE**
It is your responsibility when using Intel MKL to ensure that input data has the required format and does not contain invalid characters. These can cause unexpected behavior of the library.

| **Optimization Notice** |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. |

| **Optimization Notice** |
| --- |
| Notice revision #20110804 |

# What's New

This Developer Guide documents Intel® Math Kernel Library (Intel® MKL) 11.3 Update 2.

---

**NOTE**
This publication, the *Intel Math Kernel Library Developer Guide*, was previously known as the *Intel Math Kernel Library User's Guide*.

---

The Developer Guide has been updated with the following changes to the product and document enhancements:

- More of Intel MKL has been threaded with Intel® Threading Building Blocks (Intel® TBB). For more details, see Functions Threaded with Intel Threading Building Blocks.

Additionally, minor updates have been made to fix inaccuracies in the document.

# Notational Conventions

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

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

The following notations are used to refer to Intel MKL directories.

| | |
|---|---|
| *<parent directory>* | The installation directory that includes Intel MKL directory; for example, the directory for Intel® Parallel Studio XE Composer Edition. |
| *<mkl directory>* | The main directory where Intel MKL is installed:<br>*<mkl directory>*=*<parent directory>*`\mkl`.<br>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:<br>see *Intel MKL Reference Manual*. |
| `Monospace lowercase mixed with uppercase` | Indicates:<br>• Commands and command-line options, for example,<br>`ifort myprog.f mkl_blas95.lib mkl_c.lib libiomp5md.lib`<br>• Filenames, directory names, and pathnames, for example,<br>`C:\Program Files\Java\jdk1.5.0_09`<br>• C/C++ code fragments, for example,<br>`a = new double [SIZE*SIZE];` |
| `UPPERCASE MONOSPACE` | Indicates system variables, for example, `$MKLPATH`. |
| *`Monospace italic`* | Indicates a parameter in discussions, for example, *`lda`*.<br>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® Math Kernel Library Reference Manual*, which provides *reference* information on routine functionalities, parameter descriptions, interfaces, calling syntaxes, and return values.
- The *Intel® Math Kernel Library for Windows* OS Release Notes*.

# Getting Started

**1**

| **Optimization Notice** |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

## Checking Your Installation

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

1. Intel MKL installs in the `<parent directory>` directory.

   Check that the subdirectory of `<parent directory>` referred to as `<mkl directory>` was created.

   Check that subdirectories for Intel MKL redistributable DLLs `redist\ia32_win\mkl` and `redist\intel64_win\mkl` were created in the `<parent directory>` directory (See `redist.txt` in the Intel MKL documentation directory for a list of files that can be redistributed.)
2. If you want to keep multiple versions of Intel MKL installed on your system, update your build scripts to point to the correct Intel MKL version.
3. Check that the `mklvars.bat` file appears in the `<mkl directory>\bin` directory.

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

### See Also

Notational Conventions

## Setting Environment Variables

When the installation of Intel MKL for Windows* OS is complete, set the `PATH`, `LIB`, and `INCLUDE` environment variables in the command shell using the `mklvars.bat` script in the `bin` subdirectory of the Intel MKL installation directory. The environment variable `MIC_LD_LIBRARY_PATH` specifies locations of shared objects for Intel® Many Integrated Core (Intel® MIC) Architecture.

The script accepts the parameters, explained in the following table:

| Setting Specified | Required (Yes/No) | Possible Values | Comment |
|---|---|---|---|
| Architecture | Yes, | `ia32` | |

| Setting Specified | Required (Yes/No) | Possible Values | Comment |
|---|---|---|---|
| | when applicable | `intel64`<br>`mic` | |
| Use of Intel MKL Fortran modules precompiled with the Intel® Visual 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
  `mklvars ia32`
  sets the environment for Intel MKL to use the IA-32 architecture.
- The command
  `mklvars intel64 mod ilp64`
  sets the environment for Intel MKL to use the Intel 64 architecture, ILP64 programming interface, and Fortran modules.
- The command
  `mklvars intel64 mod`
  sets the environment for Intel MKL to use the Intel 64 architecture, LP64 interface, and Fortran modules.
- The command
  `mklvars mic lp64`
  sets the environment for Intel MKL to use the Intel MIC Architecture and LP64 programming interface.

**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
Interface Libraries and Modules
Fortran 95 Interfaces to LAPACK and BLAS
Setting the Number of Threads Using an OpenMP* Environment Variable

# Compiler Support

Intel® MKL supports compilers identified in the *Release Notes*. However, the library has been successfully used with other compilers as well.

Although Compaq no longer supports the Compaq Visual Fortran* (CVF) compiler, Intel MKL still preserves the CVF interface in the IA-32 architecture implementation. You can use this interface with the Intel® Fortran Compiler. Intel MKL provides both stdcall (default CVF interface) and cdecl (default interface of the Microsoft Visual C* application) interfaces for the IA-32 architecture.

When building Intel MKL code examples, you can select a compiler:

- For Fortran examples: Intel® or PGI* compiler
- For C examples: Intel, Microsoft Visual C++*, or PGI compiler

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

### See Also

Compiling an Application that Calls the Intel® Math Kernel Library and Uses the CVF Calling Conventions

Using the cdecl and stdcall Interfaces

Include Files

# Using Code Examples

The Intel MKL package includes code examples, located in the `examples` subdirectory of the installation directory. Use the examples to determine:

- Whether Intel MKL is working on your system
- How you should call the library
- How to link the library

If an Intel MKL 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 MKL 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. You can find examples of Automatic Offload in the `mic_ao` subdirectory (extracted from the `examples_mic` archive) and examples of Compiler Assisted Offload in the `mic_offload` subdirectory. Source code for the examples is in the next-level `sources` subdirectory.

### See Also

High-level Directory Structure

Using Intel® Math Kernel Library on Intel® Xeon Phi™ Coprocessors

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

| | |
|---|---|
| Target platform | Identify the architecture of your target machine:<br>• IA-32 or compatible<br>• Intel® 64 or compatible<br>**Reason:** Because Intel MKL libraries are located in directories corresponding to your particular architecture (seeArchitecture Support), you should provide proper paths on your link lines (see Linking Examples). To configure your development environment for the use with Intel MKL, 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 MKL function domains that you require:<br>• BLAS<br>• Sparse BLAS<br>• LAPACK<br>• PBLAS<br>• ScaLAPACK<br>• Sparse Solver routines<br>• Parallel Direct Sparse Solvers for Clusters |

| | |
|---|---|
| | • Vector Mathematics functions (VM)<br>• Vector Statistics functions (VS)<br>• Fourier Transform functions (FFT)<br>• Cluster FFT<br>• Trigonometric Transform routines<br>• Poisson, Laplace, and Helmholtz Solver routines<br>• Optimization (Trust-Region) Solver routines<br>• Data Fitting Functions<br>• Extended Eigensolver Functions<br><br>**Reason:** The function domain you intend to use narrows the search in the *Intel MKL Reference Manual* for specific routines you need. Additionally, if you are using the Intel MKL cluster software, your link line is function-domain specific (see Working with the Intel® 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 MKL provides support for both Fortran and C/C++ programming. Identify the language interfaces that your function domains support (see Appendix A: Intel® Math Kernel Library Language Interfaces Support).<br><br>**Reason:** Intel MKL provides language-specific include files for each function domain to simplify program development (see Language Interfaces Support_ by Function Domain).<br><br>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® 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).<br><br>**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. LP64 Interface). |
| Threading model | Identify whether and how your application is threaded:<br><br>• Threaded with the Intel compiler<br>• Threaded with a third-party compiler<br>• Not threaded<br><br>**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 MKL in the sequential mode (for more information, see Linking with Threading Libraries). |
| Number of threads | If your application uses an OpenMP* threading run-time library, determine the number of threads you want Intel MKL to use.<br><br>**Reason:** By default, the OpenMP* run-time library sets the number of threads for Intel MKL. If you need a different number, you have to set it yourself using one of the available mechanisms. For more information, see Improving Performance with Threading. |
| Linking model | Decide which linking model is appropriate for linking your application with Intel MKL libraries:<br><br>• Static<br>• Dynamic<br><br>**Reason:** The link 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® Math Kernel Library. |

| | |
|---|---|
| MPI used | Decide what MPI you will use with the Intel MKL cluster software. You are strongly encouraged to use the latest available version of Intel® MPI.<br><br>**Reason:** To link your application with ScaLAPACK and/or Cluster FFT, the libraries corresponding to your particular MPI should be listed on the link line (see Working with the Intel® Math Kernel Library Cluster Software). |

# Structure of the Intel® Math Kernel Library

2

| Optimization Notice |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

## Architecture Support

Intel® Math Kernel Library (Intel® MKL) for Windows* OS provides architecture-specific implementations for supported platforms. The following table lists the supported architectures and directories where each architecture-specific implementation is located.

| Architecture | Location |
|---|---|
| IA-32 or compatible | `<mkl directory>\lib\ia32_win`<br>`<parent directory>\redist\ia32_win\mkl` (DLLs) |
| Intel® 64 or compatible | `<mkl directory>\lib\intel64_win`<br>`<parent directory>\redist\intel64_win\mkl` (DLLs) |
| Intel® Many Integrated Core (Intel® MIC) | `<mkl directory>\lib\intel64_win_mic` |

### See Also

High-level Directory Structure
Notational Conventions
Detailed Structure of the IA-32 Architecture Directories
Detailed Structure of the Intel® 64 Architecture Directories

## High-level Directory Structure

| Directory | Contents |
|---|---|
| *<mkl directory>* | Installation directory of the Intel® Math Kernel Library (Intel® MKL) |
| **Subdirectories of** *<mkl directory>* | |
| `bin` | Batch files to set environmental variables in the user shell |
| `bin\ia32` | Batch files for the IA-32 architecture |

| Directory | Contents |
|---|---|
| `bin\intel64` | Batch files for the Intel® 64 architecture |
| `benchmarks\linpack` | Shared-Memory (SMP) version of the LINPACK benchmark |
| `benchmarks\mp_linpack` | Message-passing interface (MPI) version of the LINPACK benchmark |
| `lib\ia32_win` | Static libraries and static interfaces to DLLs for the IA-32 architecture |
| `lib\intel64_win` | Static libraries and static interfaces to DLLs for the Intel® 64 architecture |
| `lib\intel64_win_mic` | Static libraries and static interfaces to DLLs for the Intel® MIC architecture |
| `examples` | Source and data files for Intel MKL examples. Provided in archives corresponding to Intel MKL components selected during installation. |
| `include` | Include files for the library routines and examples |
| `include\ia32` | Fortran 95 .mod files for the IA-32 architecture and Intel Fortran compiler |
| `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\mic\lp64` | Fortran 95 .mod files for the Intel® MIC architecture, Intel® Fortran compiler, and LP64 interface |
| `include\mic\ilp64` | Fortran 95 .mod files for the Intel® MIC 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\fftw2x_cdft` | MPI FFTW 2.x interfaces to Intel MKL Cluster FFT |
| `interfaces\fftw3x_cdft` | MPI FFTW 3.x interfaces to Intel MKL Cluster FFT |
| `interfaces\fftw2xc` | FFTW 2.x interfaces to the Intel MKL FFT (C interface) |
| `interfaces\fftw2xf` | FFTW 2.x interfaces to the Intel MKL FFT (Fortran interface) |
| `interfaces\fftw3xc` | FFTW 3.x interfaces to the Intel MKL FFT (C interface) |
| `interfaces\fftw3xf` | FFTW 3.x interfaces to the Intel MKL FFT (Fortran interface) |
| `interfaces\lapack95` | Fortran 95 interfaces to LAPACK and a makefile to build the library |
| `tools` | Command-line link tool and tools for creating custom dynamically linkable libraries |
| `tools\builder` | Tools for creating custom dynamically linkable libraries |
| **Subdirectories of** *<parent directory>* | |
| `redist\ia32_win\mkl` | DLLs for applications running on processors with the IA-32 architecture |
| `redist\intel64_win\mkl` | DLLs for applications running on processors with Intel® 64 architecture |

**See Also**

Notational Conventions

Using Code Examples

# Layered Model Concept

Intel MKL 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 MKL 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 MKL 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 MKL.

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:<br>• cdecl and CVF default interfaces.<br>• LP64 and ILP64 interfaces.<br>• Compatibility with compilers that return function values differently.<br>• A mapping between single-precision names and double-precision names for applications using Cray*-style naming (SP2DP interface).<br><br>**NOTE**<br>SP2DP interface is deprecated and may be removed in a future release. |
| Threading Layer | This layer:<br>• Provides a way to link threaded Intel MKL with supported compilers.<br>• Enables you to link with a threaded or sequential mode of the library.<br><br>This layer is compiled for different environments (threaded or sequential) and compilers (from Intel and PGI*). |
| 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
Using the SP2DP Interface
Linking Your Application with the Intel® Math Kernel Library
Linking with Threading Libraries

# Linking Your Application with the Intel® Math Kernel Library

3

**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

## Linking Quick Start

Intel® Math Kernel Library (Intel® MKL) provides several options for quick linking of your application. The simplest options depend on your development environment:

| | |
|---|---|
| Intel® Parallel Studio XE Composer Edition compiler | see Using the /Qmkl Compiler Option. |
| Microsoft Visual Studio* Integrated Development Environment (IDE) | see Automatically Linking a Project in the Visual Studio* Integrated Development Environment with Intel® MKL. |

Other options are independent of your development environment, but depend on the way you link:

| | |
|---|---|
| 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 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` or `/Qmkl:parallel` — to link with a certain Intel MKL threading layer depending on the threading option provided:

- 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 MKL. |
| `/Qmkl:cluster` | to link with Intel MKL cluster components (sequential) that use Intel MPI. |

**NOTE**
The `-qopenmp` option has higher priority than `-tbb` in choosing the Intel MKL threading layer for linking.

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

For each variant of the `/Qmkl` option, the compiler links your application using the following conventions:

- cdecl for the IA-32 architecture
- LP64 for the Intel® 64 architecture

If you specify any variant of the `/Qmkl` compiler option, the compiler automatically includes the Intel MKL libraries. In cases not covered by the option, use the Link-line Advisor or see Linking in Detail.

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

## Automatically Linking a Project in the Visual Studio* Integrated Development Environment with Intel® MKL

After a default installation of the Intel® Math Kernel Library (Intel® MKL) or Intel® Parallel Studio XE Composer Edition, you can easily configure your project to automatically link with Intel MKL.

### Automatically Linking Your Microsoft Visual C/C++* Project with Intel® MKL

Configure your Microsoft Visual C/C++* project for automatic linking with Intel MKL as follows:

1. Go to **Project**>**Properties**>**Configuration Properties**>**Intel Performance Libraries**.
2. Change the **Use MKL** property setting by selecting **Parallel**, **Sequential**, or **Cluster** as appropriate.

Specific Intel MKL libraries that link with your application may depend on more project settings. For details, see the documentation for Intel®Parallel Studio XE Composer Edition for C++.

### See Also
Intel® Software Documentation Library for the documentation for Intel® Parallel Studio XE Composer Edition

### Automatically Linking Your Intel® Visual Fortran Project with Intel® MKL

Configure your Intel® Visual Fortran project for automatic linking with Intel MKL as follows:

Go to **Project** > **Properties** > **Libraries** > **Use Intel Math Kernel Library** and select **Parallel**, **Sequential**, or **Cluster** as appropriate.

Specific Intel MKL libraries that link with your application may depend on more project settings. For details see the documentation for Intel® Parallel Studio XE Composer Edition for Fortran.

### See Also

Intel® Software Documentation Library for the documentation for Intel® Parallel Studio XE Composer Edition

## Using the Single Dynamic Library

You can simplify your link line through the use of the Intel MKL Single Dynamic Library (SDL).

To use SDL, place `mkl_rt.lib` on your link line. For example:

```
icl.exe application.c mkl_rt.lib
```

`mkl_rt.lib` is the import library for `mkl_rt.dll`.

SDL enables you to select the interface and threading library for Intel MKL 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 MKL, you need to specify your choices using functions or environment variables as explained in section Dynamically Selecting the Interface and Threading Layer.

## Selecting Libraries to Link with

To link with Intel MKL:

- 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 MKL libraries to link with your application.

| | **Interface layer** | **Threading layer** | **Computational layer** | **RTL** |
|---|---|---|---|---|
| **IA-32 architecture, static linking** | `mkl_intel_c.lib` | `mkl_intel_thread.lib` | `mkl_core.lib` | `libiomp5md.lib` |
| **IA-32 architecture, dynamic linking** | `mkl_intel_c_dll.lib` | `mkl_intel_thread_dll.lib` | `mkl_core_dll.lib` | `libiomp5md.lib` |
| **Intel® 64 architecture, static linking** | `mkl_intel_lp64.lib` | `mkl_intel_thread.lib` | `mkl_core.lib` | `libiomp5md.lib` |
| **Intel® 64 architecture, dynamic linking** | `mkl_intel_lp64_dll.lib` | `mkl_intel_thread_dll.lib` | `mkl_core_dll.lib` | `libiomp5md.lib` |
| **Intel® Many Integrated Core (Intel® MIC) Architecture, static linking** | `libmkl_intel_lp64.a` | `libmkl_intel_thread.a` | `libmkl_core.a` | `libiomp5.so` |
| **Intel MIC Architecture, dynamic linking** | `libmkl_intel_lp64.so` | `libmkl_intel_thread.so` | `libmkl_core.so` | `libiomp5.so` |

The Single Dynamic Library (SDL) automatically links interface, threading, and computational libraries and thus simplifies linking. The following table lists Intel MKL libraries for dynamic linking using SDL. See Dynamically 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 |
|---|---|---|
| **IA-32 and Intel® 64 architectures** | `mkl_rt.lib` | `libiomp5md.lib`† |

†Linking with `libiomp5md.lib` is not required.

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 /Qmkl Compiler Option
Working with the Intel® Math Kernel Library Cluster Software

## Using the Link-line Advisor

Use the Intel MKL 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 http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor. 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 MKL (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 MKL to simplify building your application with Intel MKL.

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.exe` is installed in the `<mkl directory>\tools` directory.

See the knowledge base article at http://software.intel.com/en-us/articles/mkl-command-line-link-tool for more information.

# Linking Examples

### See Also

Using the Link-line Advisor
Examples of Linking for Clusters

## Linking on IA-32 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 `icl`:

- Static linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the cdecl interface:

  ```
  ifort myprog.f mkl_intel_c.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib
  ```
- Dynamic linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the cdecl interface:

  ```
  ifort myprog.f mkl_intel_c_dll.lib mkl_intel_thread_dll.lib mkl_core_dll.lib
  libiomp5md.lib
  ```
- Static linking of `myprog.f` and sequential version of Intel MKL supporting the cdecl interface:

  ```
  ifort myprog.f mkl_intel_c.lib mkl_sequential.lib mkl_core.lib
  ```
- Dynamic linking of `myprog.f` and sequential version of Intel MKL supporting the cdecl interface:

  ```
  ifort myprog.f mkl_intel_c_dll.lib mkl_sequential_dll.lib mkl_core_dll.lib
  ```
- Static linking of user code `myprog.f` and OpenMP* threaded Intel MKL supporting the stdcall interface:

  ```
  ifort myprog.f mkl_intel_s.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib
  ```
- Dynamic linking of user code `myprog.f` and OpenMP* threaded Intel MKL supporting the stdcall interface:

  ```
  ifort myprog.f mkl_intel_s_dll.lib mkl_intel_thread_dll.lib mkl_core_dll.lib
  libiomp5md.lib
  ```
- Dynamic linking of `myprog.f` and OpenMP* threaded or sequential Intel MKL supporting the cdecl or stdcall interface (Call the `mkl_set_threading_layer` function or set value of the `MKL_THREADING_LAYER` environment variable to choose threaded or sequential mode):

  ```
  ifort myprog.f mkl_rt.lib
  ```
- Static linking of `myprog.f`, Fortran 95 LAPACK interface, and OpenMP* threaded Intel MKL supporting the cdecl interface:

  ```
  ifort myprog.f mkl_lapack95.lib mkl_intel_c.lib mkl_intel_thread.lib mkl_core.lib
  libiomp5md.lib
  ```
- Static linking of `myprog.f`, Fortran 95 BLAS interface, and OpenMP* threaded Intel MKL supporting the cdecl interface:

  ```
  ifort myprog.f mkl_blas95.lib mkl_intel_c.lib mkl_intel_thread.lib mkl_core.lib
  libiomp5md.lib
  ```
- Static linking of `myprog.c` and Intel MKL threaded with Intel® Threading Building Blocks (Intel® TBB), provided that the `LIB` environment variable contains the path to Intel TBB library:

  ```
  icl myprog.c /link /libpath:$MKLPATH -I$MKLINCLUDE mkl_intel.lib mkl_tbb_thread.lib
  mkl_core.lib tbb.lib
  ```
- Dynamic linking of `myprog.c` and Intel MKL threaded with Intel TBB, provided that the `LIB` environment variable contains the path to Intel TBB library:

  ```
  icl myprog.c /link /libpath:$MKLPATH -I$MKLINCLUDE mkl_intel_dll.lib
  mkl_tbb_thread_dll.lib mkl_core_dll.lib tbb.lib
  ```

### See Also

Fortran 95 Interfaces to LAPACK and BLAS
Examples for Linking a C Application
Examples for Linking a Fortran Application
Using the Single Dynamic Library

## 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 `icl`:

- Static linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the LP64 interface:

  ```
  ifort myprog.f mkl_intel_lp64.lib mkl_intel_thread.lib mkl_core.lib
  ```

```
libiomp5md.lib
```

- Dynamic linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the LP64 interface:

  ```
  ifort myprog.f mkl_intel_lp64_dll.lib mkl_intel_thread_dll.lib mkl_core_dll.lib
  libiomp5md.lib
  ```
- Static linking of `myprog.f` and sequential version of Intel MKL supporting the LP64 interface:

  ```
  ifort myprog.f mkl_intel_lp64.lib mkl_sequential.lib mkl_core.lib
  ```
- Dynamic linking of `myprog.f` and sequential version of Intel MKL supporting the LP64 interface:

  ```
  ifort myprog.f mkl_intel_lp64_dll.lib mkl_sequential_dll.lib mkl_core_dll.lib
  ```
- Static linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the ILP64 interface:

  ```
  ifort myprog.f mkl_intel_ilp64.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib
  ```
- Dynamic linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the ILP64 interface:

  ```
  ifort myprog.f mkl_intel_ilp64_dll.lib mkl_intel_thread_dll.lib mkl_core_dll.lib
  libiomp5md.lib
  ```
- Dynamic linking of user code `myprog.f` and OpenMP* threaded or sequential Intel MKL supporting the LP64 or ILP64 interface (Call appropriate functions or set environment variables to choose threaded or sequential mode and to set the interface):

  ```
  ifort myprog.f mkl_rt.lib
  ```
- Static linking of `myprog.f`, Fortran 95 LAPACK interface, and OpenMP* threaded Intel MKL supporting the LP64 interface:

  ```
  ifort myprog.f mkl_lapack95_lp64.lib mkl_intel_lp64.lib mkl_intel_thread.lib
  mkl_core.lib libiomp5md.lib
  ```
- Static linking of `myprog.f`, Fortran 95 BLAS interface, and OpenMP* threaded Intel MKL supporting the LP64 interface:

  ```
  ifort myprog.f mkl_blas95_lp64.lib mkl_intel_lp64.lib mkl_intel_thread.lib
  mkl_core.lib libiomp5md.lib
  ```
- Static linking of `myprog.c` and Intel MKL threaded with Intel® Threading Building Blocks (Intel® TBB), provided that the `LIB` environment variable contains the path to Intel TBB library:

  ```
  icl myprog.c /link /libpath:%MKLPATH% -I%MKLINCLUDE% mkl_intel_lp64.lib
  mkl_tbb_thread.lib mkl_core.lib tbb.lib
  ```
- Dynamic linking of `myprog.c` and Intel MKL threaded with Intel TBB, provided that the `LIB` environment variable contains the path to Intel TBB library:

  ```
  icl myprog.c /link /libpath:%MKLPATH% -I%MKLINCLUDE% mkl_intel_lp64_dll.lib
  mkl_tbb_thread_dll.lib mkl_core_dll.lib tbb.lib
  ```

### See Also

Fortran 95 Interfaces to LAPACK and BLAS
Examples for Linking a C Application
Examples for Linking a Fortran Application
Using the Single Dynamic Library

# Linking in Detail

This section recommends which libraries to link with depending on your Intel MKL usage scenario and provides details of the linking.

## Dynamically Selecting the Interface and Threading Layer

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

## 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.

See the *Intel MKL Reference Manual* for details of the `mkl_set_interface_layer` function.

On systems based on the IA-32 architecture, the cdecl and stdcall interfaces are available. These interfaces have different function naming conventions, and SDL selects between cdecl and stdcall at link time according to the function names.

## 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 MKL | SEQUENTIAL | MKL_THREADING_SEQUENTIAL |
| PGI threading† | PGI | MKL_THREADING_PGI |
| Intel TBB threading | TBB | MKL_THREADING_TBB |

† Not supported by the SDL for Intel® Many Integrated Core Architecture.

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

See the *Intel MKL Reference Manual* for details of the `mkl_set_threading_layer` function.

## Replacing Error Handling and Progress Information Routines

You can replace the Intel MKL error handling routine `xerbla` or progress information routine `mkl_progress` with your own function. If you are using SDL, to replace `xerbla` or `mkl_progress`, call the `mkl_set_xerbla` and `mkl_set_progress` function, respectively. See the *Intel MKL Reference Manual* for details.

**NOTE**
If you are using SDL, you cannot perform the replacement by linking the object file with your implementation of `xerbla` or `mkl_progress`.

**See Also**
Using the Single Dynamic Library
Layered Model Concept
Using the cdecl and stdcall Interfaces
Directory Structure in Detail

## Linking with Interface Libraries

### Using the cdecl and stdcall Interfaces

cdecl and stdcall calling conventions differ in the way how the stack is restored after a function call. Intel MKL supports both conventions in its IA-32 architecture implementation through the `mkl_intel_c[_dll].lib` and `mkl_intel_s[_dll].lib` interface libraries. These libraries assume the defaults of different compilers, which also differ in the position of the string lengths in the lists of parameters passed to the calling program, as explained in the following table:

| Library for Static Linking | Library for Dynamic Linking | Calling Convention | Position of String Lengths in Parameter Lists |
|---|---|---|---|
| `mkl_intel_c.lib` | `mkl_intel_c_dll.lib` | cdecl | At the end |
| | | The defaults of Intel® C++ and Intel® Fortran compilers | |
| `mkl_intel_s.lib` | `mkl_intel_s_dll.lib` | stdcall | Immediately after the string address |
| | | The defaults of Compaq Visual Fortran* (CVF) compiler | |

**Important**
To avoid errors, ensure that the calling and called programs use the same calling convention.

To use the cdecl or stdcall calling convention, use appropriate calling syntax in C applications and appropriate compiler options for Fortran applications.

If you are using a C compiler, to link with the cdecl or stdcall interface library, call Intel MKL routines in your code as explained in the table below:

| Interface Library | Calling Intel MKL Routines |
|---|---|
| `mkl_intel_c [_dll].lib` | Use the following declaration:<br>`<type> name( <prototype variable1>, <prototype  variable2>, .. );` |
| `mkl_intel_s [_dll].lib` | Call a routine with the following statement:<br>`extern __stdcall name( <prototype variable1>, <prototype variable2>, .. );` |

If you are using a Fortran compiler, to link with the cdecl or stdcall interface library, provide compiler options as explained in the table below:

| Interface Library | Compiler Options | Comment |
|---|---|---|
| | **Intel® Fortran compiler** | |
| `mkl_intel_c[_dll].lib` | Default | |
| `mkl_intel_s[_dll].lib` | `/Gm`<br>or<br>`/iface:cvf` | `/Gm` and `/iface:cvf` options enable compatibility of the CVF and Powerstation calling conventions |
| | **CVF compiler** | |
| `mkl_intel_s[_dll].lib` | Default | |
| `mkl_intel_c[_dll].lib` | `/iface=(cref, nomixed_str_len_arg)` | |

### See Also

Using the stdcall Calling Convention in C/C++
Compiling an Application that Calls the Intel® Math Kernel Library and Uses the CVF Calling Conventions

## Using the ILP64 Interface vs. LP64 Interface

The Intel MKL 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:

- `mkl_intel_lp64.lib` or `mkl_intel_ilp64.lib` for static linking
- `mkl_intel_lp64_dll.lib` or `mkl_intel_ilp64_dll.lib` 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 `/4I8` compiler option

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

Intel MKL 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 /4I8 /I`<*mkl directory*>`\include ...` |
| Compiling for LP64 | `ifort /I`<*mkl directory*>`\include ...` |

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

**CAUTION**
Linking of an application compiled with the `/4I8` 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 MKL 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 MKL 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 MKL Reference Manual* for more information.

To better understand ILP64 interface details, see also examples.

## Limitations

All Intel MKL function domains support ILP64 programming but FFTW interfaces to Intel MKL:

- 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
Include Files

Language Interfaces Support, by Function Domain
Layered Model Concept
Directory Structure in Detail

### Using the SP2DP Interface

> **NOTE**
> SP2DP Interface is deprecated and may be removed in a future release.

If your application is targeted to the Intel 64 architecture and uses Cray*-style naming and ILP64 interface, SP2DP interface enables you to call Intel MKL BLAS and/or LAPACK functions from your application with minimal changes to the code.

SP2DP interface maps single-precision names (for both real and complex types) in the application to double-precision names in Intel MKL BLAS and LAPACK. Function names are mapped as shown in the following example for BLAS functions `?GEMM`:

```
SGEMM -> DGEMM

DGEMM -> DGEMM

CGEMM -> ZGEMM

ZGEMM -> ZGEMM
```

> **NOTE**
> No changes are made to double-precision names.

The SP2DP interface is implemented in the Interface layer. To use the SP2DP interface, statically link your application with the `mkl_intel_sp2dp.lib` library.

For example, use the following link line to statically link your Fortran application with the threaded version of Intel MKL using the SP2DP interface:

```
ifort myprog.f mkl_intel_sp2dp.lib mkl_intel_thread.lib mkl_core.lib

libiomp5md.lib
```

#### See Also

High-level Directory Structure
Layered Model Concept
Directory Structure in Detail

### Linking with Fortran 95 Interface Libraries

The `mkl_blas95*.lib` and `mkl_lapack95*.lib` libraries contain Fortran 95 interfaces for BLAS and LAPACK, respectively, which are compiler-dependent. In the Intel MKL 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 MKL threading layer defines how Intel MKL functions utilize multiple computing cores of the system that the application runs on. You must link your application with one appropriate Intel MKL library in this layer, as explained below. Depending on whether this is a threading or a sequential library, Intel MKL runs in a parallel or sequential mode, respectively.

In the *parallel mode*, Intel MKL utilizes multiple processor cores available on your system, uses the OpenMP* or Intel TBB threading technology, and requires a proper threading run-time library (RTL) to be linked with your application. Independently of use of Intel MKL, 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 MKL provides several threading libraries, each dependent on the threading RTL of a certain compiler, and your choice of the Intel MKL threading library must be consistent with the threading RTL that you use in your application.

The OpenMP RTL of the Intel® compiler is the `libiomp5md.lib` library, located under `<parent directory>\compiler\lib`. 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 `tbb.lib` 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 MKL 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 MKL threading libraries is compatible with
- Your application is already threaded at a top level, and using parallel Intel MKL 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 MKL threading layer and what threading RTL you should choose under different scenarios:

| **Application** | | **Intel MKL** | | **RTL Required** |
|---|---|---|---|---|
| **Uses OpenMP** | **Compiled with** | **Execution Mode** | **Threading Layer** | |
| no | any compiler | parallel | Static linking:<br>`mkl_intel_thread.lib`<br>Dynamic linking:<br>`mkl_intel_thread_dll.lib` | `libiomp5md.lib` |
| no | any compiler | parallel | Static linking:<br>`mkl_tbb_thread.lib`<br>Dynamic linking:<br>`mkl_tbb_thread_dll.lib` | `tbb.lib` |
| no | any compiler | sequential | Static linking:<br>`mkl_sequential.lib`<br>Dynamic linking:<br>`mkl_sequential_dll.lib` | none |
| yes | Intel compiler | parallel | Static linking:<br>`mkl_intel_thread.lib`<br>Dynamic linking: | `libiomp5md.lib` |

| Application | | Intel MKL | | RTL Required |
|---|---|---|---|---|
| **Uses OpenMP** | **Compiled with** | **Execution Mode** | **Threading Layer** | |
| yes | PGI* compiler | parallel | `mkl_intel_thread_dll.lib`<br>Static linking:<br>`mkl_pgi_thread.lib`<br>Dynamic linking:<br>`mkl_pgi_thread_dll.lib` | PGI OpenMP RTL |
| yes | any other compiler | parallel | Not supported. Use Intel MKL in the sequential mode. | |

**See Also**
Layered Model Concept
Notational Conventions

## Linking with Computational Libraries

If you are not using the Intel MKL ScaLAPACK and Cluster Fast Fourier Transforms (FFT), you need to link your application with only one computational library, depending on the linking method:

| Static Linking | Dynamic Linking |
|---|---|
| `mkl_core.lib` | `mkl_core_dll.lib` |

### Computational Libraries for Applications that Use ScaLAPACK or Cluster FFT

ScaLAPACK and Cluster FFT require more computational libraries, which may depend on your architecture.

The following table lists computational libraries for IA -32 architecture applications that use ScaLAPACK or Cluster FFT.

**Computational Libraries for IA-32 Architecture**

| Function domain | Static Linking | Dynamic Linking |
|---|---|---|
| ScaLAPACK † | `mkl_scalapack_core.lib`<br>`mkl_core.lib` | `mkl_scalapack_core_dll.lib`<br>`mkl_core_dll.lib` |
| Cluster Fourier Transform Functions† | `mkl_cdft_core.lib`<br>`mkl_core.lib` | `mkl_cdft_core_dll.lib`<br>`mkl_core_dll.lib` |

† Also add the library with BLACS routines corresponding to the MPI used.

The following table lists computational libraries for Intel® 64 or Intel® Many Integrated Core Architecture applications that use ScaLAPACK or Cluster FFT.

**Computational Libraries for the Intel® 64 or Intel® Many Integrated Core Architecture**

| Function domain | Static Linking | Dynamic Linking |
|---|---|---|
| ScaLAPACK, LP64 interface‡ | `mkl_scalapack_lp64.lib`<br>`mkl_core.lib` | `mkl_scalapack_lp64_dll.lib`<br>`mkl_core_dll.lib` |
| ScaLAPACK, ILP64 interface‡ | `mkl_scalapack_ilp64.lib`<br>`mkl_core.lib` | `mkl_scalapack_ilp64_dll.lib`<br>`mkl_core_dll.lib` |
| Cluster Fourier Transform Functions‡ | `mkl_cdft_core.lib`<br>`mkl_core.lib` | `mkl_cdft_core_dll.lib`<br>`mkl_core_dll.lib` |

‡ Also add the library with BLACS routines corresponding to the MPI used.

### See Also
Linking with Intel MKL Cluster Software
Using the Link-line Advisor
Using the ILP64 Interface vs. LP64 Interface

## Linking with Compiler Run-time Libraries

Dynamically link `libiomp5` or `tbb` 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 `tbb` dynamically, be sure the `PATH` environment variable is defined correctly.

Sometimes you may improve performance of your application with threaded Intel MKL by using the `/MT` compiler option. The compiler driver will pass the option to the linker and the latter will load multi-thread (MT) static run-time libraries.

However, to link a Vector Mathematics (VM) application that uses the *errno* variable for error reporting, compile and link your code using the option that depends on the linking model:

- `/MT` for linking with static Intel MKL libraries
- `/MD` for linking with dynamic Intel MKL libraries

### See Also
Setting Environment Variables
Layered Model Concept

## Linking with System Libraries

If your system is based on the Intel® 64architecture, be aware that Microsoft SDK builds 1289 or higher provide the `bufferoverflowu.lib` library to resolve the `__security_cookie` external references. Makefiles for examples and tests include this library by using the `buf_lib=bufferoverflowu.lib` macro. If you are using older SDKs, leave this macro empty on your command line as follows: `buf_lib=` .

### See Also
Linking Examples

# Building Custom Dynamic-link Libraries

Custom dynamic-link libraries (DLL) reduce the collection of functions available in Intel MKL 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 MKL custom DLL builder enables you to create a dynamic 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 Dynamic-link Library Builder in the Command-line Mode

To build a custom DLL, use the following command:

`nmake target [<options>]`

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

| Value | Comment |
|---|---|
| `libia32` | The builder uses static Intel MKL interface, threading, and core libraries to build a custom DLL for the IA-32 architecture. |
| `libintel64` | The builder uses static Intel MKL interface, threading, and core libraries to build a custom DLL for the Intel® 64 architecture. |
| `dllia32` | The builder uses the single dynamic library `libmkl_rt.dll` to build a custom DLL for the IA-32 architecture. |
| `dllintel64` | The builder uses the single dynamic library `libmkl_rt.dll` to build a custom DLL for the Intel® 64 architecture. |
| `help` | The command prints Help on the custom DLL 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` | Defines which programming interface to use.Possible values:<br>• For the IA-32 architecture, `{cdecl|stdcall}`. The default value is `cdecl`.<br>• For the Intel 64 architecture, `{lp64|ilp64}`. The default value is `lp64`. |
| `threading = {parallel|sequential}` | Defines whether to use the Intel MKL in the threaded or sequential mode. The default value is `parallel`. |
| `export = <file name>` | Specifies the full name of the file that contains the list of entry-point functions to be included in the DLL. The default name is `user_example_list` (no extension). |
| `name = <dll name>` | Specifies the name of the dll and interface library to be created. By default, the names of the created libraries are `mkl_custom.dll` and `mkl_custom.lib`. |
| `xerbla = <error handler>` | Specifies the name of the object file *`<user_xerbla>`*`.obj` that contains the user's error handler. The makefile adds this error handler to the library for use instead of the default Intel MKL error handler *`xerbla`*. If you omit this parameter, the native Intel MKL *`xerbla`* is used. See the description of the *`xerbla`* function in the Intel MKL Reference Manual on how to develop your own error handler. For the IA-32 architecture, the object file should be in the interface defined by the interface macro (cdecl or stdcall). |

| Parameter [Values] | Description |
|---|---|
| `MKLROOT = <mkl directory>` | Specifies the location of Intel MKL libraries used to build the custom DLL. By default, the builder uses the Intel MKL installation directory. |
| `buf_lib` | Manages resolution of the `__security_cookie` external references in the custom DLL on systems based on the Intel® 64 architecture.<br>By default, the makefile uses the `bufferoverflowu.lib` library of Microsoft SDK builds 1289 or higher. This library resolves the `__security_cookie` external references.<br>To avoid using this library, set the empty value of this parameter. Therefore, if you are using an older SDK, set `buf_lib=` .<br>**CAUTION**<br>Use the `buf_lib` parameter only with the empty value. Incorrect value of the parameter causes builder errors. |
| `crt = <c run-time library>` | Specifies the name of the Microsoft C run-time library to be used to build the custom DLL. By default, the builder uses `msvcrt.lib`. |
| `manifest = {yes\|no\|embed}` | Manages the creation of a Microsoft manifest for the custom DLL:<br>• If `manifest=yes`, the manifest file with the name defined by the name parameter above and the `manifest` extension will be created.<br>• If `manifest=no`, the manifest file will not be created.<br>• If `manifest=embed`, the manifest will be embedded into the DLL.<br>By default, the builder does not use the `manifest` parameter. |

All the above parameters are optional.

In the simplest case, the command line is `nmake ia32`, and the missing options have default values. This command creates the `mkl_custom.dll` and `mkl_custom.lib` libraries with the cdecl interface for processors using the IA-32 architecture. The command takes the list of functions from the `functions_list` file and uses the native Intel MKL error handler *`xerbla`*.

An example of a more complex case follows:

```
nmake ia32 interface=stdcall export=my_func_list.txt name=mkl_smallxerbla=my_xerbla.obj
```

In this case, the command creates the `mkl_small.dll` and `mkl_small.lib` libraries with the stdcall interface for processors using the IA-32 architecture. The command takes the list of functions from `my_func_list.txt` file and uses the user's error handler `my_xerbla.obj`.

The process is similar for processors using the Intel® 64 architecture.

### See Also

Linking with System Libraries

## Composing a List of Functions

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

1. Link your application with installed Intel MKL libraries to make sure the application builds.
2. Remove all Intel MKL libraries from the link line and start linking.

   Unresolved symbols indicate Intel MKL functions that your application uses.

3. Include these functions in the list.

> **Important**
> Each time your application starts using more Intel MKL 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 DLL, adjust function names to the required interface. For example, you can list the cdecl entry points as follows:

```
DGEMM

DTRSM

DDOT

DGETRF

DGETRS

cblas_dgemm

cblas_ddot
```

You can list the stdcall entry points as follows:

```
_DGEMM@60

_DDOT@20

_DGETRF@24
```

For more examples, see domain-specific lists of function names in the `<mkl directory>\tools\builder` folder. This folder contains lists of function names for both cdecl or stdcall interfaces.

> **NOTE**
> The lists of function names 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 DLL.

> **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.

## Building a Custom Dynamic-link Library in the Visual Studio* Development System

You can build a custom dynamic-link library (DLL) in the Microsoft Visual Studio* Development System (VS*). To do this, use projects available in the `tools\builder\MSVS_Projects` subdirectory of the Intel MKL directory. The directory contains subdirectories with projects for the respective versions of the Visual Studio Development System, for example, `VS2012`. For each version of VS two solutions are available:

- `libia32.sln` builds a custom DLL for the IA-32 architecture.
- `libintel64.sln` builds a custom DLL for the Intel® 64 architecture.

The builder uses the following default settings for the custom DLL:

| | |
|---|---|
| **Interface:** | cdecl for the IA-32 architecture and LP64 for the Intel 64 architecture |
| **Error handler:** | Native Intel MKL `xerbla` |
| **Create Microsoft manifest:** | yes |
| **List of functions:** | in the project's source file `examples.def` |

To build a custom DLL:

1. Set the `MKLROOT` environment variable with the installation directory of the Intel MKL version you are going to use.
2. Open the `libia32.sln` or `libintel64.sln` solution depending on the architecture of your system.

   The solution includes the following projects:

   - `i_malloc_dll`
   - `vml_dll_core`
   - `cdecl_parallel` (in `libia32.sln`) or `lp64_parallel` (in `libintel64.sln`)
   - `cdecl_sequential` (in `libia32.sln`) or `lp64_sequential` (in `libintel64.sln`)
3. [Optional] To change any of the default settings, select the project depending on whether the DLL will use Intel MKL functions in the sequential or multi-threaded mode:

   - In the `libia32` solution, select the `cdecl_sequential` or `cdecl_parallel` project.
   - In the `libintel64` solution, select the `lp64_sequential` or `lp64_parallel` project.
4. [Optional] To build the DLL that uses the stdcall interface for the IA-32 architecture or the ILP64 interface for the Intel 64 architecture:

   a. Select **Project**>**Properties**>**Configuration Properties**>**Linker**>**Input**>**Additional Dependencies**.
   b. In the `libia32` solution, change `mkl_intel_c.lib` to `mkl_intel_s.lib`.
   In the `libintel64` solution, change `mkl_intel_lp64.lib` to `mkl_intel_ilp64.lib`.
5. [Optional] To include your own error handler in the DLL:

   a. Select **Project**>**Properties**>**Configuration Properties**>**Linker**>**Input**.
   b. Add `<user_xerbla>.obj`
6. [Optional] To turn off creation of the manifest:

   a. Select **Project**>**Properties**>**Configuration Properties**>**Linker**>**Manifest File**>**Generate Manifest**.
   b. Select: no.
7. [Optional] To change the list of functions to be included in the DLL:

   a. Select **Source Files**.
   b. Edit the `examples.def` file. Refer to Specifying Function Names for how to specify entry points.
8. To build the library, select **Build**>**Build Solution**.

**See Also**

Using the Custom Dynamic-link Library Builder in the Command-line Mode

## Distributing Your Custom Dynamic-link Library

To enable use of your custom DLL in a threaded mode, distribute `libiomp5md.dll` along with the custom DLL.

# Managing Performance and Memory

# 4

| Optimization Notice |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

## Improving Performance with Threading

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

Intel MKL is *thread-safe*, which means that all Intel MKL functions (except the LAPACK deprecated routine `?lacon`) work correctly during simultaneous execution by multiple threads. In particular, any chunk of threaded Intel MKL 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 MKL 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 MKL also offers variables that are independent of OpenMP, such as `MKL_NUM_THREADS`, and equivalent Intel MKL functions for thread management. The Intel MKL 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 MKL uses the number of OpenMP 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.

### See Also

Managing Multi-core Performance

### OpenMP* Threaded Functions and Problems

The following Intel MKL 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* for Intel® Core™2 Duo and Intel® Core™ i7 processors:

- Level1 BLAS:
  `?axpy`, `?copy`, `?swap`, `ddot/sdot`, `cdotc`, `drot/srot`
- Level2 BLAS:
  `?gemv`, `?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. |
| IA-32 | *N* is a power of 2, $log_2(N) > 13$, and the transform is single-precision. |
| | *N* is a power of 2, $log_2(N) > 14$, and the transform is double-precision. |
| 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 MKL function domains are threaded with Intel® Threading Building Blocks (Intel® TBB):

- LAPACK:

  For a list of threaded routines, see LAPACK Routines.
- Entire Level3 BLAS
- Fast Poisson, Laplace, and Helmholtz Solver (Poisson Library)
- Intel MKL PARDISO, a direct sparse solver based on Parallel Direct Sparse Solver (PARDISO*).

  For details, see Intel MKL PARDISO Steps.
- Sparse BLAS

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

### LAPACK Routines

The following LAPACK routines are threaded with Intel TBB:
`?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 Intel TBB threading:
`?getrs`, `?gesv`, `?potrs`, `?bdsqr`, and `?gels`.

### Intel MKL PARDISO Steps

Reordering and factorization steps of the solver are threaded with Intel TBB. In the solving step, call the routines sequentially.

### Sparse BLAS Routines

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

## Avoiding Conflicts in the Execution Environment

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

If your program is parallelized by other means than Intel® OpenMP* run-time library (RTL) and Intel TBB RTL, several calls to Intel MKL 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 Intel TBB (for example: Win32* threads on Windows* OS). | If more than one thread calls Intel MKL, and the function being called is threaded, it may be important that you turn off Intel MKL threading. Set the number of threads to one by any of the available means (see Techniques 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 MKL threading library that matches the compiler you use (see Linking Examples on how to do this). If this is not possible, use Intel MKL in the sequential mode. To do this, you should link with the appropriate threading library: `mkl_sequential.lib` or `mkl_sequential.dll` (see Appendix C: Directory Structure in Detail). |
| You thread the program using Intel TBB 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 MKL TBB threading library and Intel TBB RTL if it matches the compiler you use. If this is not possible, use Intel MKL in the sequential mode. To do this, link with the appropriate threading library: `mkl_sequential.lib` or `mkl_sequential_dll.lib` (see Appendix C: Directory Structure in Detail). |
| You run multiple programs calling Intel MKL 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). Section Intel® Optimized MP LINPACK Benchmark for Clusters discusses another solution for a Hybrid (OpenMP* + MPI) mode. |

Using the `mkl_set_num_threads` and `mkl_domain_set_num_threads` functions to control parallelism of Intel MKL 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 MKL Reference Manual* for the function description).

### See Also

Using Additional Threading Control
Linking with Compiler Run-time Libraries

## Techniques to Set the Number of Threads

Use the following techniques to specify the number of OpenMP threads to use in Intel MKL:

- Set one of the OpenMP or Intel MKL environment variables:
  - `OMP_NUM_THREADS`
  - `MKL_NUM_THREADS`

- `MKL_DOMAIN_NUM_THREADS`
- Call one of the OpenMP or Intel MKL 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 MKL and may result in slow Intel MKL 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 MKL Reference Manual* for the function description).

---

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

- The Intel MKL 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 MKL 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 MKL.

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:

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

Some shells require the variable and its value to be exported:

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

You can alternatively assign value to the environment variable using Microsoft Windows*OS Control Panel.

Note that you will not benefit from setting this variable on Microsoft Windows* 98 orWindows* ME because multiprocessing is not supported.

### 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 );`

## Using Additional Threading Control

### Intel MKL-specific Environment Variables for OpenMP Threading Control

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

> **NOTE**
> Some Intel MKL 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.

Section "Number of User Threads" in the "Fourier Transform Functions" chapter of the *Intel MKL Reference Manual* shows how the Intel MKL threading controls help to set the number of threads for the FFT computation.

The table below lists the Intel MKL 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`<br>`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 MKL to dynamically change the number of OpenMP threads. | `OMP_DYNAMIC` |

> **NOTE**
> Call `mkl_set_num_threads()` to force Intel MKL to use a given number of OpenMP threads and prevent it from reacting to the environment variables `MKL_NUM_THREADS`, `MKL_DOMAIN_NUM_THREADS`, and `OMP_NUM_THREADS`.

The example below shows how to force Intel MKL to use one thread:

See the *Intel MKL Reference Manual* for the detailed description of the threading control functions, their parameters, calling syntax, and more code examples.

## MKL_DYNAMIC

The `MKL_DYNAMIC` environment variable enables Intel MKL 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 MKL 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 MKL 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 (it is impossible to detect this with MPICH 1.2.x, for instance), Intel MKL runs one OpenMP thread.

When `MKL_DYNAMIC` is `FALSE`, Intel MKL 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 MKL is called from an OpenMP parallel region in your program, Intel MKL uses only one thread by default. If you want Intel MKL to go parallel in such a call, link your program against an OpenMP threading RTL supported by Intel MKL 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 MKL uses `MKL_NUM_THREADS` threads when it is called from the OpenMP parallel region in your program.

In general, set `MKL_DYNAMIC` to `FALSE` only under circumstances that Intel MKL 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 | non-cluster Fourier Transform Functions |
| MKL_DOMAIN_VML | Vector Mathematics (VM) |
| MKL_DOMAIN_PARDISO | Intel MKL PARDISO, a direct sparse solver based on Parallel Direct Sparse Solver (PARDISO*) |

For example,

```
MKL_DOMAIN_ALL 2 : MKL_DOMAIN_BLAS 1 : MKL_DOMAIN_FFT 4
MKL_DOMAIN_ALL=2 : MKL_DOMAIN_BLAS=1 : MKL_DOMAIN_FFT=4
MKL_DOMAIN_ALL=2,  MKL_DOMAIN_BLAS=1,  MKL_DOMAIN_FFT=4
MKL_DOMAIN_ALL=2;  MKL_DOMAIN_BLAS=1;  MKL_DOMAIN_FFT=4
MKL_DOMAIN_ALL  = 2  MKL_DOMAIN_BLAS 1 ,  MKL_DOMAIN_FFT  4
MKL_DOMAIN_ALL,2: MKL_DOMAIN_BLAS 1, MKL_DOMAIN_FFT,4 .
```

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 MKL threading control functions, can be found in the mkl.h header file.

The table below 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 MKL 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 MKL 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 MKL. |

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 );
```

## 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:

```
set <VARIABLE NAME>=<value>
```

For example:

```
set MKL_NUM_THREADS=4
set MKL_DOMAIN_NUM_THREADS="MKL_DOMAIN_ALL=1, MKL_DOMAIN_BLAS=4"
set MKL_DYNAMIC=FALSE
```

Some shells require the variable and its value to be exported:

```
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
```

You can alternatively assign values to the environment variables using Microsoft Windows* OS Control Panel.

# Calling Intel MKL Functions from Multi-threaded Applications

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

## Usage model: disable Intel MKL internal threading for the whole application

**When used:** Intel MKL 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.lib` 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.lib` 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 MKL.
- Set the `MKL_DOMAIN_NUM_THREADS` environment variable to a desired number of OpenMP threads for Intel MKL for a particular function domain.

  Use if the application threads work with different Intel MKL function domains.
- Call `mkl_set_num_threads()`

  Use to globally set a desired number of OpenMP threads for Intel MKL at run time.
- Call `mkl_domain_set_num_threads()`.

  Use if at some point application threads start working with different Intel MKL function domains.
- Call `mkl_set_num_threads_local()`.

  Use to set the number of OpenMP threads for Intel MKL 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 MKL 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 MKL Reference Manual*, available in the Intel Software Documentation Library.

| **Optimization Notice** |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

## See Also

Linking with Threading Libraries
Dynamically Selecting the Interface and Threading Layer
Intel MKL-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 MKL 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 MKL, apply the following setting:

```
set 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.

## Managing Multi-core Performance

You can obtain best performance on systems with multi-core processors by requiring thatthreads do not migrate from core to core. To do this, bind threads to the CPU cores bysetting an affinity mask to threads. Use one of the following options:

- OpenMP facilities (if available), for example, the`KMP_AFFINITY`environment variable using the Intel OpenMP library
- A system function, as explained below
- Intel TBB facilities (if available), for example, the `tbb::affinity_partitioner` class (for details, see https://www.threadingbuildingblocks.org/documentation)

Consider the following performance issue:

- The system has two sockets with two cores each, for a total of four cores (CPUs).
- The application sets the number of OpenMP threads to four and calls an Intel MKL LAPACK routine. This call takes considerably different amounts of time from run to run.

To resolve this issue, before calling Intel MKL, set an affinity mask for each OpenMP thread using the `KMP_AFFINITY` environment variable or the `SetThreadAffinityMask` system function. The following code example shows how to resolve the issue by setting an affinity mask by operating system means using the Intel compiler. The code calls the function`SetThreadAffinityMask` to bind the threads toappropriatecores,preventing migration of the threads. Then the Intel MKLLAPACK routineis called:

Compile the application with the Intel compiler using the following command:

```
icl /Qopenmp test_application.c
```

where`test_application.c`is the filename for the application.

Build the application. Run it in four threads, for example, by using the environment variable to set the number of threads:

```
set OMP_NUM_THREADS=4
test_application.exe
```

See Windows API documentation at msdn.microsoft.com/ for the restrictions on the usage of Windows API routines and particulars of the `SetThreadAffinityMask` function used in the above example.

See also a similar example at en.wikipedia.org/wiki/Affinity_mask.

# Improving Performance for Small Size Problems

The overhead of calling an Intel MKL function for small problem sizes can be significant when the function has a large number of parameters or internally checks parameter errors. To reduce the performance overhead for these small size problems, the Intel MKL *direct call* feature works in conjunction with the compiler to preprocess the calling parameters to `?gemm` functions and directly call or inline special optimized small-matrix kernels that bypass error checking.

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 MKL is required by supplying the compiler option as explained below:

| Intel MKL 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 and `-Mpreprocess` option for PGI* compilers.
  - Include the Intel MKL Fortran include file `mkl_direct_call.fi`.

Intel MKL skips error checking and intermediate function calls if the problem size is small enough (for example, a `dgemm` function call with matrix ranks smaller than 50).

| Optimization Notice |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. |
| Notice revision #20110804 |

## Using MKL_DIRECT_CALL in C Applications

The following examples of code and link lines show how to activate direct calls to Intel MKL kernels in C applications:

- Include the `mkl.h` header file:
- For multi-threaded Intel MKL, compile with `MKL_DIRECT_CALL` preprocessor macro:

```
icl /DMKL_DIRECT_CALL /Qstd=c99 your_application.c mkl_intel_lp64.lib mkl_core.lib
mkl_intel_thread.lib /Qopenmp -I%MKLROOT%/include
```

- To use Intel MKL in the sequential mode, compile with `MKL_DIRECT_CALL_SEQ` preprocessor macro:

```
icl /DMKL_DIRECT_CALL_SEQ /Qstd=c99 your_application.c mkl_intel_lp64.lib mkl_core.lib
mkl_sequential.lib -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 MKL kernels in Fortran applications:

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

- For multi-threaded Intel MKL, compile with /`fpp` option for Intel Fortran compiler (or with `-Mpreprocess` for PGI compilers) and with `MKL_DIRECT_CALL` preprocessor macro:

```
ifort /DMKL_DIRECT_CALL /fpp your_application.f mkl_intel_lp64.lib mkl_core.lib mkl_intel_thread.lib
/Qopenmp -I%MKLROOT%/include
```

- To use Intel MKL 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 mkl_intel_lp64.lib mkl_core.lib
mkl_sequential.lib
-I%MKLROOT%/include
```

### Limitations of the Direct Call

Directly calling the Intel MKL kernels has the following limitations:

- If the `MKL_DIRECT_CALL` or `MKL_DIRECT_CALL_SEQ` macro is used, Intel MKL 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 `?gemm`, `?gemm3m`, `?syrk`, `?trsm`, `?axpy`, and `?dot` functions.
- Intel MKL 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 `/Qstd=c99` option for Intel® compilers.
- In a fixed format Fortran source code compiled with PGI compilers, the lines containing Intel MKL functions must end at least seven columns before the line ending column, usually, in a column with the index not greater than 72 – 7 = 65.

> **NOTE**
> The direct call feature substitutes the names of Intel MKL functions with longer counterparts, which can cause the lines to exceed the column limit for a fixed format Fortran source code compiled with PGI compilers. Because the compilers ignore any part of the line that exceeds the limit, the behavior of the program can be unpredictable.

# Other Tips and Techniques to Improve Performance

### See Also

Managing Performance of the Cluster Fourier Transform Functions
Improving Performance on Intel Xeon Phi Coprocessors Tips for Intel® Many Integrated Core Architecture

## Coding Techniques

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

### Data Alignment

To improve performance of your application that calls Intel MKL, align your arrays on 64-byte boundaries and ensure that the leading dimensions of the arrays are divisible by 64. 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 MKL Reference Manual). 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:

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

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

#### See Also

Example of Data Alignment
Improving Performance on Intel Xeon Phi Coprocessors
Managing Performance of the Cluster Fourier Transform Functions

## Improving Intel(R) MKL Performance on Specific Processors

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

To get the best performance with Intel MKL 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 MKL 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.

## Fast Fourier Transform Optimized Radices

You can improve the performance of Intel MKL Fourier Transform Functions if the length of your data vector permits factorization into powers of optimized radices.

In Intel MKL, the optimized radices are 2, 3, 5, 7, 11, and 13.

# Using Memory Functions

## Avoiding Memory Leaks in Intel MKL

When running, Intel MKL allocates and deallocates internal buffers to facilitate better performance. However, in some cases this behavior may result in memory leaks.

To avoid memory leaks, you can do either of the following:

- Set the `MKL_DISABLE_FAST_MM` environment variable to 1 or call the `mkl_disable_fast_mm()` function.

  Be aware that this change may negatively impact performance of some Intel MKL functions, especially for small problem sizes.
- Call the `mkl_free_buffers()` function or the `mkl_thread_free_buffers()` function in the current thread.

For the descriptions of the memory functions, see the Intel MKL Reference Manual, available in the Intel Software Documentation Library.

### See Also

Intel Software Documentation Library

## Redefining Memory Functions

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

### Memory Renaming

Intel MKL 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 MKL 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:

If you areusing the statically linked Intel MKL,

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 MKL functions, as shown in the following example:

If you are using the dynamically linked Intel MKL,

1. Include the `i_malloc.h` header file in your code.
2. Redefine values of pointers `i_malloc_dll, i_free_dll, i_calloc_dll`, and `i_realloc_dll` prior to the first call to MKL functions, as shown in the following example:

# Language-specific Usage Options

# 5

The Intel® Math Kernel Library (Intel® MKL) 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 MKL.

| **Optimization Notice** |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

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

This section discusses mixed-language programming and the use of language-specific interfaces with Intel MKL.

See also the "FFTW Interface to Intel® Math Kernel Library" Appendix in the Intel MKL Reference Manual for details of the FFTW interfaces to Intel MKL.

### 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 MKL architecture-specific directories** | |
| `mkl_blas95.lib`[1] | Fortran 95 wrappers for BLAS (BLAS95) for IA-32 architecture. |
| `mkl_blas95_ilp64.lib`[1] | Fortran 95 wrappers for BLAS (BLAS95) supporting LP64 interface. |
| `mkl_blas95_lp64.lib`[1] | Fortran 95 wrappers for BLAS (BLAS95) supporting ILP64 interface. |
| `mkl_lapack95.lib`[1] | Fortran 95 wrappers for LAPACK (LAPACK95) for IA-32 architecture. |
| `mkl_lapack95_lp64.lib`[1] | Fortran 95 wrappers for LAPACK (LAPACK95) supporting LP64 interface. |

| File name | Contains |
|---|---|
| mkl_lapack95_ilp64.lib[1] | Fortran 95 wrappers for LAPACK (LAPACK95) supporting ILP64 interface. |
| fftw2xc_intel.lib[1] | Interfaces for FFTW version 2.x (C interface for Intel compilers) to call Intel MKL FFT. |
| fftw2xc_ms.lib | Contains interfaces for FFTW version 2.x (C interface for Microsoft compilers) to call Intel MKL FFT. |
| fftw2xf_intel.lib | Interfaces for FFTW version 2.x (Fortran interface for Intel compilers) to call Intel MKL FFT. |
| fftw3xc_intel.lib[2] | Interfaces for FFTW version 3.x (C interface for Intel compiler) to call Intel MKL FFT. |
| fftw3xc_ms.lib | Interfaces for FFTW version 3.x (C interface for Microsoft compilers) to call Intel MKL FFT. |
| fftw3xf_intel.lib[2] | Interfaces for FFTW version 3.x (Fortran interface for Intel compilers) to call Intel MKL FFT. |
| fftw2x_cdft_SINGLE.lib | Single-precision interfaces for MPI FFTW version 2.x (C interface) to call Intel MKL cluster FFT. |
| fftw2x_cdft_DOUBLE.lib | Double-precision interfaces for MPI FFTW version 2.x (C interface) to call Intel MKL cluster FFT. |
| fftw3x_cdft.lib | Interfaces for MPI FFTW version 3.x (C interface) to call Intel MKL cluster FFT. |
| fftw3x_cdft_ilp64.lib | Interfaces for MPI FFTW version 3.x (C interface) to call Intel MKL cluster FFT supporting the ILP64 interface. |
| **Modules, in architecture- and interface-specific subdirectories of the Intel MKL 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 MKL support functions. |

[1] Prebuilt for the Intel® Fortran compiler

[2] FFTW3 interfaces are integrated with Intel MKL. 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 MKL 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, see Compiler-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 one of the following commands depending on your architecture:

   - For the IA-32 architecture,

     `nmake libia32 install_dir=<user dir>`

   - For the Intel® 64 architecture,

     `nmake 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 one of `{ia32, intel64}`.

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

For example, the command

`nmake libintel64 FC=f95 install_dir=<userf95 dir> interface=lp64`

builds the required library and `.mod` files and installs them in subdirectories of `<userf95 dir>`.

To delete the library from the building directory, use one of the following commands:

- For the IA-32 architecture,

  `nmake cleania32 install_dir=<user dir>`

- For the Intel® 64 architecture,

  `nmake cleanintel64 [interface=lp64|ilp64] install_dir=<user dir>`

- For all the architectures,

  `nmake 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 MKL 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 MKL 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 MKL delivers these modules compiled with the Intel compiler, along with source code, to be used with different compilers.

## Using the stdcall Calling Convention in C/C++

Intel MKL supports stdcall calling convention for the following function domains:

- BLAS, except CBLAS
- Sparse BLAS
- LAPACK
- Vector Mathematics
- Vector Statistics (VS)
- Intel MKL PARDISO,
  a direct sparse solver based on Parallel Direct Sparse Solver (PARDISO*)
- Direct Sparse Solvers
- RCI Iterative Solvers
- Support Functions

To use the stdcall calling convention in C/C++, follow the guidelines below:

- In your function calls, pass lengths of character strings to the functions. For example, compare the following calls to the VS function `vslLoadStreamF`:

  cdecl: `errstatus = vslLoadStreamF(&stream, "streamfile.bin");`

  stdcall: `errstatus = vslLoadStreamF(&stream, "streamfile.bin", 14);`
- Define the `MKL_STDCALL` macro using either of the following techniques:
  - Define the macro in your source code before including Intel MKL header files:
  - Pass the macro to the compiler. For example:

    `icl -DMKL_STDCALL foo.c`
- Link your application with the following library:
  - `mkl_intel_s.lib` for static linking
  - `mkl_intel_s_dll.lib` for dynamic linking

---

**See Also**

Using the cdecl and stdcall Interfaces

Compiling an Application that Calls the Intel® Math Kernel Library and Uses the CVF Calling Conventions

Include Files

## Compiling an Application that Calls the Intel® Math Kernel Library and Uses the CVF Calling Conventions

The IA-32 architecture implementation of Intel MKL supports the Compaq Visual Fortran* (CVF) calling convention by providing the stdcall interface.

Although the Intel MKL does not provide the CVF interface in its Intel® 64 architecture implementation, you can use the Intel® Visual Fortran Compiler to compile your Intel® 64 architecture application that calls Intel MKL and uses the CVF calling convention. To do this:

- Provide the following compiler options to enable compatibility with the CVF calling convention:
  `/Gm` or `/iface:cvf`
- Additionally provide the following options to enable calling Intel MKL from your application:
  `/iface:nomixed_str_len_arg`

**See Also**

Using the cdecl and stdcall Interfaces

Compiler Support

# Mixed-language Programming with the Intel Math Kernel Library

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

See also these Knowledge Base articles:

- http://software.intel.com/en-us/articles/performance-tools-for-software-developers-how-do-i-use-intel-mkl-with-java for how to call Intel MKL from Java* applications.
- http://software.intel.com/en-us/articles/how-to-use-boost-ublas-with-intel-mkl for how to perform BLAS matrix-matrix multiplication in C++ using Intel MKL substitution of Boost* uBLAS functions.
- http://software.intel.com/en-us/articles/intel-mkl-and-third-party-applications-how-to-use-them-together for a list of articles describing how to use Intel MKL with third-party libraries and applications.

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

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

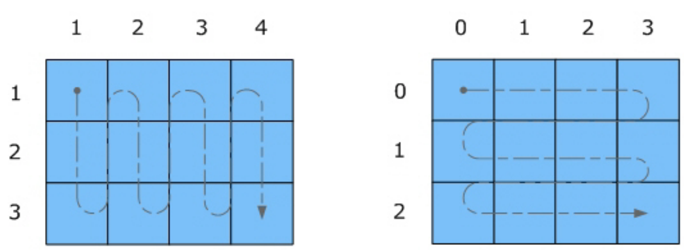
> **CAUTION**
> Avoid calling BLAS 95/LAPACK 95 from C/C++. Such calls require skills in manipulating the descriptor of a deferred-shape array, which is the Fortran 90 type. Moreover, BLAS95/LAPACK95 routines contain links to a Fortran RTL.

### 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(R) MKL Reference Manual 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. The header file 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 MKL.

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 MKL installation directory.

## Using Complex Types in C/C++

As described in the documentation for the Intel®Visual 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 MKL 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:

and

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®Visual 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 MKL 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`.

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

Below is the C++ implementation:

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

This example uses CBLAS:

# Obtaining Numerically Reproducible Results

# 6

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

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 MKL does run-time processor dispatching in order to identify the appropriate internal code paths to traverse for the Intel MKL 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 MKL 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 MKL are designated by the latest ISA they use for optimizations: from the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) to the Intel® Advanced Vector Extensions 2 (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 MKL. So it is natural that consistent results require some performance trade-offs. If limited to a particular code path, performance of Intel MKL 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.

| Optimization Notice |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor- |

| **Optimization Notice** |
|---|
| dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

# Getting Started with Conditional Numerical Reproducibility

Intel MKL offers functions and environment variables to help you get reproducible results. You can configure Intel MKL 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 MKL functions running in the CNR mode, but refer to Reproducibility Conditions for details related to data alignment.

## Intel CPUs supporting Intel AVX2

To ensure Intel MKL 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 MKL function calls
3. Do either of the following:

   - Call

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

     `set 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 MKL calls return the same results on every Intel CPU 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 MKL function calls
3. Do either of the following:

   - Call

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

     `set 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 MKL 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 MKL function calls
3. Do either of the following:

   - Call

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

     `set 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 MKL 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 MKL Reference Manual*:

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

### See Also
Code Examples

# Specifying the Code Branches

Intel MKL provides conditional numerically reproducible results for a code branch determined by the supported instruction set architecture (ISA). The values you can specify for the `MKL_CBWR` environment variable may have one of the following equivalent formats:

- `MKL_CBWR="<branch>"`
- `MKL_CBWR="BRANCH=<branch>"`

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 |

| Value | Description |
|---|---|
| | **CNR mode uses the branch for the following ISA:** |
| `COMPATIBLE` | Intel® Streaming SIMD Extensions 2 (Intel® SSE2) without rcpps/ rsqrtps instructions |
| `SSE2` | Intel SSE2 |
| `SSE3` | DEPRECATED. Intel® Streaming SIMD Extensions 3 (Intel® SSE3). This setting is kept for backward compatibility and is equivalent to `SSE2`. |
| `SSSE3` | Supplemental Streaming SIMD Extensions 3 (SSSE3) |
| `SSE4_1` | Intel® Streaming SIMD Extensions 4-1 (Intel® SSE4-1) |
| `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) |

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.

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

- 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 MKL Reference Manual* 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.

- 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 Intel MKL 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 sets Intel MKL to run in the CNR mode based on the default dispatching for your platform. The other two commands are equivalent and set the CNR branch to Intel AVX:

- `set MKL_CBWR=AUTO`
- `set MKL_CBWR=AVX`
- `set MKL_CBWR=BRANCH=AVX`

**See Also**

Specifying the Code Branches

# Code Examples

The following simple programs show how to obtain reproducible results from run to run of Intel MKL functions. See the *Intel MKL Reference Manual* for more examples.

**C Example of CNR**

**Fortran Example of CNR**

**Use of CNR with Unaligned Data in C**

**Use of CNR with Unaligned Data in Fortran**

# Coding Tips

# 7

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

## Example of Data Alignment

Needs for best performance with Intel MKL or for reproducible results from run to run of Intel MKL 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

## 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 MKL major version |
| `__INTEL_MKL_MINOR__` | Intel MKL minor version |
| `__INTEL_MKL_UPDATE__` | Intel MKL update number |
| `INTEL_MKL_VERSION` | Intel MKL full version in the following format:<br>`INTEL_MKL_VERSION` =<br>(`__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`,<br>or `mkl.h`, which includes `mkl_version.h` |
| Intel®Fortran compiler: | `mkl.fi` |
| Any Fortran compiler with enabled preprocessing: | `mkl_version.h`<br>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 MKL. In this case, the version is 11.2 Update 4:

**Intel®Fortran Compiler:**

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

# Managing Output

8

## Using Intel MKL Verbose Mode

If your application calls Intel MKL functions, you may want to know what computational functions are called, what parameters are passed to them, and how much time is spent to execute the functions. Your application can print this information to a standard output device if Intel MKL Verbose mode is enabled. Functions that can print this information are referred to as *verbose-enabled* functions. While not all Intel MKL functions are verbose-enabled, see *Intel MKL Release Notes* for the Intel MKL function domains that support the Verbose mode.

In the Verbose mode, every call of a verbose-enabled function finishes with printing a human readable line describing the call. If the application is terminated during the function call, no information for that function is printed. The first call to a verbose-enabled function also prints a version information line.

To enable the Intel MKL Verbose mode for an application, do one of the following:

- Set the environment variable `MKL_VERBOSE` to 1.
- Call the support function `mkl_verbose`(1).

The function call `mkl_verbose`(0) disables the Verbose mode. Enabling or disabling the Verbose mode using the function call takes precedence over the environment setting. For a full description of the `mkl_verbose` function, see the *Intel MKL Reference Manual*, available in the Intel® Software Documentation Library.

Intel MKL Verbose mode is not a thread-local but a global state. It means that 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 MKL 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 MKL 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 | |

| Information | Description | Related Links |
|---|---|---|
| | • `OSX` for OS X* | |
| The host CPU frequency. | | |
| Intel MKL interface layer used by the application. | Possible values:<br>• `stdcall` or `cdecl` on systems based on the IA-32 architecture.<br>• `lp64` or `ilp64` on systems based on the Intel® 64 architecture. | Using the cdecl and stdcall Interfaces<br>Using the ILP64 Interface vs. LP64 Interface |
| Intel MKL threading layer used by the application. | Possible values:<br>`intel_thread`, `tbb_thread`, `pgi_thread`, or `sequential`. | Linking with Threading Libraries |
| The number of Intel® Xeon Phi™ coprocessors detected. | Nothing is printed if no coprocessors are detected.<br>The number printed is prefixed with `NMICDev:` .<br>Intel MKL attempts to detect the coprocessors unless it runs in the sequential mode because Automatic Offload functionality is only provided by threaded Intel MKL. | Automatic Offload |

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, Win 3.10GHz lp64
intel_thread NMICDev:2
```

## Call Description Line

In Intel MKL 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 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. | |

| Information | Description | Related Links |
|---|---|---|
| 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. To reduce this effect, bind threads that call Intel MKL to CPU cores by setting an affinity mask. | Managing Multi-core Performance for options to set an affinity mask. |
| 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` |
| Status of the Intel MKL memory manager. | The value printed is prefixed with `FastMM:` | Avoiding Memory Leaks in Intel MKL for a description of the Intel MKL memory manager |
| OpenMP* thread number of the calling thread. | The value printed is prefixed with `TID:` | |
| Values of Intel MKL environment variables defining the general and domain-specific numbers of threads, separated by a comma. | The first value printed is prefixed with `NThr:` | Intel MKL-specific Environment Variables for Threading Control |
| Value of the `MKL_HOST_WORKDIVISION` environment variable. | The value printed is prefixed with `WDiv:HOST:` | Automatic Offload Controls |
| Values of the `MKL_MIC_`*`<number>`*`_WORKDIVISION` environment variables for each Intel® Xeon Phi™ coprocessor available on the system, where *`<number>`* is the number of the coprocessor. | The first value printed is prefixed with `WDiv:`*`<number>`*`:` | Automatic Offload Controls |

The following is an example of a call description line:

```
MKL_VERBOSE DGEMM(n,n,
1000,1000,240,0x7ffff708bb30,0x7ff2aea4c000,1000,0x7ff28e92b000,240,0x7ffff708bb38,0x7f
f28e08d000,1000) 1.66ms CNR:OFF Dyn:1 FastMM:1 TID:0 NThr:16,FFT:2 WDiv:HOST:-1.000
WDiv:0:-1.000 WDiv:1:-1.000
```

The following information is not printed because of limitations of Intel MKL 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.

# Working with the Intel® Math Kernel Library Cluster Software

9

Intel® Math Kernel Library (Intel® MKL) 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 and Intel® Many Integrated Core architectures.

**Important**
ScaLAPACK, Cluster FFT, and Cluster Sparse Solver function domains are not installed by default. To use them, explicitly select the appropriate component during installation.

| Optimization Notice |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

## Message-Passing Interface Support

Intel MKL ScaLAPACK, Cluster FFT, and Cluster Sparse Solver support implementations of the message-passing interface (MPI) identified in the *Intel® Math Kernel Library (Intel® MKL) Release Notes*.

To link applications with ScaLAPACK, Cluster FFT, or Cluster Sparse Solver, you need to configure your system depending on your message-passing interface (MPI) implementation as explained below.

If you are using MPICH2, do the following:

1. Add `mpich2\include` to the include path (assuming the default MPICH2 installation).
2. Add `mpich2\lib` to the library path.
3. Add `mpi.lib` to your link command.
4. Add `fmpich2.lib` to your Fortran link command.
5. Add `cxx.lib` to your Release target link command and `cxxd.lib` to your Debug target link command for C++ programs.

If you are using the Microsoft MPI, do the following:

1. Add `Microsoft Compute Cluster Pack\include` to the include path (assuming the default installation of the Microsoft MPI).
2. Add `Microsoft Compute Cluster Pack\Lib\AMD64` to the library path.
3. Add `msmpi.lib` to your link command.

If you are using the Intel® MPI, do the following:

1. Add the following string to the include path: `%ProgramFiles%\Intel\MPI\<ver>\intel64\include`, where *<ver>* is the directory for a particular MPI version, for example, `%ProgramFiles%\Intel\MPI\5.1\intel64\include`.
2. Add the following string to the library path: `%ProgramFiles%\Intel\MPI\<ver>\intel64\lib`, for example, `%ProgramFiles%\Intel\MPI\5.1\intel64\lib`.
3. Add `impi.lib` and `impicxx.lib` to your link command.

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

# Linking with Intel MKL Cluster Software

The Intel MKL ScaLAPACK, Cluster FFT, and Cluster Sparse Solver support MPI implementations identified in the *Intel MKL Release Notes*.

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

```
set lib =<path to MKL libraries>;<path to MPI libraries>;%lib%
```

```
<linker> <files to link> [<MKL cluster library>] <BLACS><MKL core libraries><MPI libraries>
```

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

| | |
|---|---|
| *<path to MKL libraries>* | *<mkl directory>*`\lib\intel64_win`. If you performed the Scripts to Set Environment Variables Setting Environment Variables step of the Getting Started process, you do not need to add this directory to the `lib` environment variable. |
| *<path to MPI libraries>* | Typically the `lib` subdirectory in the MPI installation directory. |
| *<linker>* | One of `icl, ifort, xilink`. |
| *<MKL cluster library>* | One of libraries for ScaLAPACK or Cluster FFT listed in Appendix C: Directory Structure in Detail. For example, for the LP64 interface, it is `mkl_scalapack_lp64.lib` or `mkl_cdft_core.lib`. Cluster Sparse Solver does not require an additional computation library. |
| *<BLACS>* | The BLACS library corresponding to your , programming interface (LP64 or ILP64), and MPI version. These libraries are listed in Appendix C: Directory Structure in Detail. For example, for the LP64 interface, choose one of `mkl_blacs_intelmpi_lp64.lib`, `mkl_blacs_mpich2_lp64.lib`, or `mkl_blacs_msmpi_lp64.lib` in the case of static linking and `mkl_blacs_lp64_dll.lib` in the case of dynamic linking. |
| *<MKL core libraries>* | Intel MKL libraries other than libraries with ScaLAPACK, Cluster FFT, or Cluster Sparse Solver. |

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

Intel MPI provides prepackaged scripts for its linkers to help you link using the respective linker. Therefore, if you are using Intel MPI, the best way to link is to use the following commands:

*<path to Intel MPI binaries>*`\mpivars.bat`

`set lib =` *<path to MKL libraries>*`;%lib%`

*<mpilinker><files to link>* `[`*<MKL cluster Library>*`]` *<BLACS><MKL core libraries>*

where the placeholders that are not yet defined are explained in the following table:

| | |
|---|---|
| *<path to MPI binaries>* | By default, the `bin` subdirectory in the MPI installation directory. |
| *<MPI linker>* | `mpicl` or `mpiifort` |

**See Also**

Linking Your Application with the Intel® Math Kernel Library

Examples of Linking for Clusters

# Determining the Number of OpenMP* Threads

The OpenMP* run-time library responds to the environment variable `OMP_NUM_THREADS`. Intel MKL 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 MKL 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 (`mkl_intel_thread.lib`), 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.

The `OMP_NUM_THREADS` environment variable is assumed in the discussion below.

Set `OMP_NUM_THREADS` so that the product of its value and the number of MPI ranks per node equals the number of real processors or cores of a node. If the Intel® Hyper-Threading Technology is enabled on the node, use only half number of the processors that are visible on Windows OS.

> **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**

Setting Environment Variables on a Cluster

# Using DLLs

All the needed DLLs must be visible on all the nodes at run time, and you should install Intel® Math Kernel Library (Intel® MKL) on each node of the cluster. You can use Remote Installation Services (RIS) provided by Microsoft to remotely install the library on each of the nodes that are part of your cluster. The best way to make the DLLs visible is to point to these libraries in the `PATH` environment variable. See Setting Environment Variables on a Cluster on how to set the value of the PATH environment variable.

The ScaLAPACK DLLs in the `<parent directory>\redist\intel64_win\mkl`directory use the MPI dispatching mechanism. MPI dispatching is based on the `MKL_BLACS_MPI` environment variable. The BLACS DLL uses MKL_BLACS_MPI for choosing the needed MPI libraries. The table below lists possible values of the variable.

| Value | Comment |
|---|---|
| `MPICH2` | Default value. MPICH2 for Windows* OS is used for message passing |
| `INTELMPI` | Intel MPI is used for message passing |
| `MSMPI` | Microsoft MPI is used for message passing |
| `CUSTOM` | Intel MKL MPI wrappers built with a custom MPI are used for message passing |

If you are using a non-default MPI, assign the same appropriate value to `MKL_BLACS_MPI` on all nodes.

### See Also

Setting Environment Variables on a Cluster
Notational Conventions

# Setting Environment Variables on a Cluster

If you are using MPICH2 or higher or Intel MPI, to set an environment variable on the cluster, use `-env`, `-genv`, `-genvlist` keys of `mpiexec`.

See the following MPICH2 examples on how to set the value of `OMP_NUM_THREADS`:

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

```
mpiexec -genvlist OMP_NUM_THREADS ....
```

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

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

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

```
mpiexec -genvlist MKL_BLACS_MPI ....
```

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

When using MPICH2 or higher, you may have problems with getting the global environment, such as `MKL_BLACS_MPI`, by the `-genvlist` key. In this case, set up user or system environments on each node as follows:
From the **Start** menu, select **Settings** > **Control Panel** > **System** > **Advanced** > **Environment Variables**.

If you are using Microsoft MPI, the above ways of setting environment variables are also applicable if the Microsoft Single Program Multiple Data (SPMD) process managers are running in a debug mode on all nodes of the cluster. However, the best way to set environment variables is using the Job Scheduler with the Microsoft Management Console (MMC) and/or the Command Line Interface (CLI) to submit a job and pass environment variables. For more information about MMC and CLI, see the Microsoft Help and Support page at the Microsoft Web site (http://www.microsoft.com/).

# Interaction with the Message-passing Interface

To improve performance of cluster applications, it is critical for Intel MKL 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 in Techniques to Set the Number of Threads.

If the number of threads is not set, Intel MKL checks whether it runs under MPI provided by the Intel® MPI Library. If this is true, the following environment variables define Intel MKL 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 MKL 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 MKL conclusion of the thread support level:

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

  In all these cases, Intel MKL 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 MKL.

| **Optimization Notice** |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br><br>Notice revision #20110804 |

### See Also

Managing Multi-core Performance
Intel® Software Documentation 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 MKL provides a set of prebuilt BLACS libraries that support certain MPI libraries, but this, however, does not enable use of Intel MKL with other MPI libraries. To fill this gap, Intel MKL additionally includes the MKLMPI adaptor, which provides an MPI-independent ABI to Intel MKL. The adaptor is provided as source code. To use Intel MKL 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 MKLMPI adaptor is located in the `<mkl directory>\interfaces\mklmpi` directory.

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

For example: the command

```
nmake libintel64
```

builds a static custom BLACS library `mkl_blacs_custom_lp64.lib` using the default MPI compiler on your system. 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

```
nmake help
```

### Using a Custom BLACS Library

In the case of static linking, 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 `mkl_blacs_intelmpi_lp64.lib` static library, pass `mkl_blacs_custom_lp64.lib`.

To use a dynamic custom BLACS library:

1. Link your application the same way as when you use the prebuilt BLACS library.
2. Call the `mkl_set_mpi` support function or set the `MKL_BLACS_MPI` environment variable to one of the following values:

   - `CUSTOM`

     to load a custom library with the default name `mkl_blacs_custom_lp64.dll` or `mkl_blacs_custom_ilp64.dll`, depending on whether the BLACS interface linked against your application is LP64 or ILP64.
   - *<dll_name>*

     to load the specified BLACS DLL.

   **NOTE**
   Intel MKL looks for the specified DLL either in the directory with Intel MKL dynamic libraries or in the directory with the application executable.

   For a description of the `mkl_set_mpi` function, see the *Intel MKL Reference Manual*.

### See Also

Linking with Intel MKL 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 MKL cluster function domains runs the same way as any other MPI application (refer to the documentation that comes with your MPI implementation).

For further linking examples, see the support website for Intel products at http://www.intel.com/software/products/support/.

### See Also

Directory Structure in Detail

## Examples for Linking a C Application

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

- Main module is in C.
- MPICH2 is installed in `c:\mpich2x64`.
- You are using the Intel® C++ Compiler.
- Intel MKL functions use LP64 interfaces.

To link with ScaLAPACK for a cluster of Intel® 64 architecture based systems, set the environment variable and use the link line as follows:

```
set lib=c:\mpich2x64\lib;<mkl directory>\lib\intel64_win;%lib%
```

```
icl <user files to link> mkl_scalapack_lp64.lib mkl_blacs_mpich2_lp64.lib
mkl_intel_lp64.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib mpi.lib cxx.lib
bufferoverflowu.lib
```

To link with Cluster FFT for a cluster of Intel® 64 architecture based systems, set the environment variable and use the link line as follows:

```
set lib=c:\mpich2x64\lib;<mkl directory>\lib\intel64_win;%lib%
```

```
icl <user files to link> mkl_cdft_core.lib mkl_blacs_mpich2_lp64.lib mkl_intel_lp64.lib
mkl_intel_thread.lib mkl_core.lib libiomp5md.lib mpi.lib cxx.lib bufferoverflowu.lib
```

To link with Cluster Sparse Solver for a cluster of Intel® 64 architecture based systems, set the environment variable and use the link line as follows:

```
set lib=c:\mpich2x64\lib;<mkl directory>\lib\intel64_win;%lib%
```

```
icl <user files to link> mkl_blacs_mpich2_lp64.lib mkl_intel_lp64.lib
mkl_intel_thread.lib mkl_core.lib libiomp5md.lib mpi.lib cxx.lib bufferoverflowu.lib
```

### See Also

Linking with Intel MKL Cluster Software
Using the Link-line Advisor
Linking with System Libraries

## Examples for Linking a Fortran Application

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

- Main module is in Fortran.
- Microsoft Windows Compute Cluster Pack SDK is installed in `c:\MS CCP SDK`.
- You are using the Intel® Fortran Compiler.
- Intel MKL functions use LP64 interfaces.

To link with ScaLAPACK for a cluster of Intel® 64 architecture based systems, set the environment variable and use the link line as follows:

```
set lib="c:\MS CCP SDK\Lib\AMD64";<mkl directory>\lib\intel64_win;%lib%
```

```
ifort <user files to link> mkl_scalapack_lp64.lib mkl_blacs_mpich2_lp64.lib
mkl_intel_lp64.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib msmpi.lib
bufferoverflowu.lib
```

To link with Cluster FFT for a cluster of Intel® 64 architecture based systems, set the environment variable and use the link line as follows:

```
set lib="c:\MS CCP SDK\Lib\AMD64";<mkl directory>\lib\intel64_win;%lib%
```

```
ifort <user files to link> mkl_cdft_core.lib mkl_blacs_mpich2_lp64.lib
mkl_intel_lp64.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib msmpi.lib
bufferoverflowu.lib
```

To link with Cluster Sparse Solver for a cluster of Intel® 64 architecture based systems, set the environment variable and use the link line as follows:

```
set lib="c:\MS CCP SDK\Lib\AMD64";<mkl directory>\lib\intel64_win;%lib%
```

```
ifort <user files to link> mkl_blacs_mpich2_lp64.lib mkl_intel_lp64.lib
mkl_intel_thread.lib mkl_core.lib libiomp5md.lib msmpi.lib bufferoverflowu.lib
```

## See Also

Linking with Intel MKL Cluster Software
Using the Link-line Advisor
Linking with System Libraries

# Using Intel® Math Kernel Library on Intel® Xeon Phi™ Coprocessors

10

Intel® Math Kernel Library (Intel® MKL) offers two sets of libraries to support Intel® Many Integrated Core (Intel® MIC) Architecture:

- For the host computer based on Intel® 64 or compatible architecture and running a Windows* operating system
- For Intel® Xeon Phi™ coprocessors

You can control how Intel MKL offloads computations to Intel® Xeon Phi™ coprocessors. Either you can offload computations automatically or use Compiler Assisted Offload:

- Automatic Offload.

  On Windows* OS running on Intel® 64 or compatible architecture systems, Automatic Offload automatically detects the presence of coprocessors based on Intel MIC Architecture and automatically offloads computations that may benefit from additional computational resources available. This usage model enables you to call Intel MKL routines as you would normally do with minimal changes to your program. The only change needed to enable Automatic Offload is either the setting of an environment variable or a single function call. For details see Automatic Offload.
- Compiler Assisted Offload.

  This usage model enables you to use the Intel compiler and its offload pragma support to manage the functions and data offloaded to a coprocessor. Within an offload region, you should specify both the input and output data for the Intel MKL functions to be offloaded. After linking with the Intel MKL libraries for Intel MIC Architecture, the compiler provided run-time libraries transfer the functions along with their data to a coprocessor to carry out the computations. For details see Compiler Assisted Offload.

In addition to offloading computations to coprocessors, you can call Intel MKL functions from an application that runs natively on a coprocessor. Native execution occurs when an application runs entirely on Intel MIC Architecture. Native mode is a fast way to make an existing application run on Intel MIC Architecture with minimal changes to the source code. For more information, see Running Intel MKL on an Intel Xeon Phi Coprocessor in Native Mode.

Intel MKL ScaLAPACK and Cluster FFT can benefit from yet another usage model offered by the Intel® MPI Library. The Intel MPI Library treats each Intel Xeon Phi coprocessor as a regular node in a cluster of Intel® Xeon® processors and Intel Xeon Phi coprocessors. To run your application on a coprocessor, you can specify an MPI rank on the coprocessor, build the application for Intel MIC Architecture, and launch the built executable from the host computer or the coprocessor. For usage details of the MPI on coprocessors, see documentation for the Intel MPI Library, available in the Intel Software Documentation Library. For details of building MPI applications that use Intel MKL, see Using ScaLAPACK and Cluster FFT on Intel Xeon Phi Coprocessors.

Intel MKL functionality offers different levels of support for Intel MIC Architecture:

- Optimized
- Supported

Please see the *Intel MKL Release Notes* for details.

| **Optimization Notice** |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or |

| **Optimization Notice** |
|---|
| effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

# Automatic Offload

Automatic Offload provides performance improvements with fewer changes to the code than Compiler Assisted Offload. If you are executing a function on the host CPU, Intel MKL running in the Automatic Offload mode may offload part of the computations to one or multiple Intel Xeon Phi coprocessors without you explicitly offloading computations. By default, Intel MKL determines the best division of the work between the host CPU and coprocessors. However, you can specify a custom work division.

To enable Automatic Offload and control the division of work, use environment variables or support functions. See the *Intel MKL Reference Manual* for detailed descriptions of the support functions.

> **Important**
> Use of Automatic Offload does not require changes in your link line. However, be aware that Automatic Offload supports only OpenMP* threaded Intel MKL.

## Automatic Offload Controls

The table below lists the environment variables for Automatic Offload and the functions that cause *similar* results. See the *Intel MKL Reference Manual* for detailed descriptions of the functions. To control the division of work between the host CPU and Intel Xeon Phi coprocessors, the environment variables use a fractional measure ranging from zero to one.

| **Environment Variable** | **Support Function** | **Description** | **Value** |
|---|---|---|---|
| `MKL_MIC_ENABLE` | `mkl_mic_enable` | Enables Automatic Offload (AO). | 1 |
| `OFFLOAD_DEVICES` | None | `OFFLOAD_DEVICES` is a common setting for Intel MKL and Intel® Compilers. It specifies a list of coprocessors to be used for any offload, including Intel MKL AO.<br>In particular, this setting may help you to configure the environment for an MPI application to run Intel MKL in the AO mode.<br>If this variable is not set, all the coprocessors available on the system are used for AO. | A comma-separated list of integers, each ranging from 0 to the largest number of an Intel Xeon Phi coprocessor on the system, with the maximum of 31. Values out of this range are ignored. Moreover, if the list contains any non-integer data, the list is ignored completely as if the environment variable were not set at all.<br>For example, if your system has 4 Intel Xeon Phi coprocessors and the value of the list is 1,3, Intel MKL uses only coprocessors 1 and 3 for AO, and Intel MKL support functions and environment variables refer to these coprocessors as coprocessors 0 and 1. |

| Environment Variable | Support Function | Description | Value |
|---|---|---|---|
| | | You can set this environment variable if AO is enabled by the environment setting or function call.<br>Setting this variable to an empty value is equivalent to completely disabling AO regardless of the value of `MKL_MIC_ENABLE`.<br>After setting this environment variable, Intel MKL support functions and environment variables refer to the specified coprocessors by their indexes in the list, starting with zero.<br>For more information, refer to the *Intel® Compiler User and Reference Guides*. | |
| `OFFLOAD_ENABLE_ORSL` | None | Enables the mode in which Intel MKL and Intel Compilers synchronize their accesses to coprocessors. Set this variable if your application uses both Compiler Assisted and AO but does not implement its own synchronization. | 1 |
| `MKL_HOST_WORKDIVISION` | `mkl_mic_set_workdivision` | Specifies the fraction of work for the host CPU to do. | A floating-point number ranging from 0.0 to 1.0. For example, the value could be 0.2 or 0.33. Intel MKL ignores negative values and treats values greater than 1 as 1.0. |
| `MKL_MIC_WORKDIVISION` | `mkl_mic_set_workdivision` | Specifies the fraction of work to do on all the Intel Xeon Phi coprocessors on the system. | See `MKL_HOST_WORKDIVISION` |
| `MKL_MIC_`*<number>*`_WORKDIVISION` | `mkl_mic_set_workdivision` | Specifies the fraction of work to do on a specific Intel Xeon Phi coprocessor. Here *<number>* is an integer ranging from 0 to the largest number of an Intel Xeon Phi coprocessor on the | See `MKL_HOST_WORKDIVISION` |

| Environment Variable | Support Function | Description | Value |
|---|---|---|---|
| | | system, with the maximum of 31. For example, if the system has two Intel Xeon Phi coprocessors, *<number>* can be 0 or 1. | |
| MKL_MIC_ MAX_MEMORY | mkl_mic_set_ max_memory | Specifies the maximum coprocessor memory reserved for AO computations on all of the Intel Xeon Phi coprocessors on the system. Each process that performs AO computations uses additional coprocessor memory specified by the environment variable. | Memory size in Kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T). For example, MKL_MIC_MAX_MEMORY = 4096M limits the coprocessor memory reserved for AO computations to 4096 megabytes or 4 gigabytes. Setting MKL_MIC_MAX_MEMORY = 4G specifies the same amount of memory in gigabytes. |
| MKL_MIC_*<number>*_ MAX_MEMORY | mkl_mic_set_ max_memory | Specifies the maximum coprocessor memory reserved for AO computations on a specific Intel Xeon Phi coprocessor on the system.<br><br>Here *<number>* is an integer ranging from 0 to the largest number of an Intel Xeon Phi coprocessor on the system, with the maximum of 31. For example, if the system has two Intel Xeon Phi coprocessors, *<number>* can be 0 or 1. | Memory size in Kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T). For example, MKL_MIC_MAX_MEMORY = 4096M limits the coprocessor memory reserved for AO computations to 4096 megabytes or 4 gigabytes. Setting MKL_MIC_MAX_MEMORY = 4G specifies the same amount of memory in gigabytes. |
| MKL_MIC_ REGISTER_ MEMORY | mkl_mic_ register_ memory | Enables/disables the mkl_malloc function running in AO mode to register allocated memory.<br><br>If AO is disabled, this setting has no effect.<br><br>Setting this environment variable to 1 may improve performance if the same memory region allocated by mkl_malloc is passed multiple times to Intel MKL functions enabled for AO (for a list of AO enabled functions, see *Intel MKL Release Notes*). | Desired behavior of mkl_malloc:<br><br>0 - not register allocated memory<br><br>1 - register allocated memory |

| Environment Variable | Support Function | Description | Value |
|---|---|---|---|
| `MKL_MIC_RESOURCE_LIMIT` | `mkl_mic_set_resource_limit` | Specifies how much of the computational resources of Intel Xeon Phi coprocessors can be used by the calling process. Use this environment variable if you need to share Intel Xeon Phi coprocessor cores automatically across multiple processes that call Intel MKL in the AO mode. For example, this might be useful in MPI applications.<br><br>Actual reservation is made during a call to an Intel MKL AO function. | A floating-point number ranging from 0.0 to 1.0.<br><br>Special values:<br>• 0.0 - do not share Intel Xeon Phi coprocessors across multiple processes. Default.<br>• `MKL_MPI_PPN` - enable special fully automated mode for MPI applications. In this mode Intel MKL tries to read the number of MPI processes per node (*ppn*) from the environment variables passed to the process by MPI. If this attempt is successful, Intel MKL sets the actual resource limit to 1.0/*ppn*.<br><br>**NOTE**<br>For Intel® MPI Library, Open MPI, and IBM Platform MPI, Intel MKL automatically detects *ppn*. For other MPI implementations, use the `MKL_MPI_PPN` environment variable to set *ppn*.<br><br>Examples:<br>**1.** Two 61-core Intel Xeon Phi coprocessors are available on the system. Three processes simultaneously set `MKL_MIC_RESOURCE_LIMIT=0.34` and then call `dpotrf` in AO mode. As a result, one process receives 40 cores from coprocessor 1, another process receives 40 cores from coprocessor 2, and the remaining process receives cores from each of the two coprocessors for a total of 40 cores.<br>**2.** Two 58-core Intel Xeon Phi coprocessors are available on the system. The user makes these settings<br>`MKL_MIC_1_WORKDIVISION=0.0`<br>`MKL_MIC_PPN=4`<br>`MKL_MIC_RESOURCE_LIMIT=MKL_MPI_PPN` |

| Environment Variable | Support Function | Description | Value |
|---|---|---|---|
| | | | and then runs an MPI application with Intel MKL AO calls. As a result, each MPI process receives 14 cores on Intel Xeon Phi coprocessor 0 because the user excluded coprocessor 1 from computations by setting zero workdivision for it. |
| MIC_OMP_ NUM_THREADS | mkl_mic_set_ device_num_ threads | Specifies the maximum number of OpenMP* threads to use for AO computations on all the Intel Xeon Phi coprocessors on the system. | An integer greater than 0. |
| MIC_*<number>*_ OMP_NUM_ THREADS | mkl_mic_set_ device_num_ threads | Specifies the maximum number of OpenMP threads to use for AO computations on a specific Intel Xeon Phi coprocessor on the system.<br>Here *<number>* is an integer ranging from 0 to the largest number of an Intel Xeon Phi coprocessor on the system, with the maximum of 31. For example, if the system has two Intel Xeon Phi coprocessors, *<number>* can be 0 or 1. | An integer greater than 0. |
| OFFLOAD_ REPORT | mkl_mic_set_ offload_report | OFFLOAD_REPORT is a common setting for Intel MKL and Intel® Compilers. It specifies the profiling report level for any offload, including Intel MKL AO.<br>For more information, refer to the *Intel® Compiler User and Reference Guides*.<br>Note that the mkl_mic_set_offload_report function enables you to turn profile reporting on/off at run time but does not change the reporting level. | An integer ranging from 0 to 2:<br>0 - No reporting, default.<br>1 - The report includes:<br>• The name of the function called in the AO mode.<br>• Effective work division. The value of -1 indicates that the hint, that is, the work division specified by the mkl_mic_set_workdivison function or the appropriate MKL_*_WORKDIVISION environment variable was ignored in this function call.<br>• The time spent on the host CPU during the call.<br>• The time spent on each available Intel Xeon Phi coprocessor during the call. |

| Environment Variable | Support Function | Description | Value |
|---|---|---|---|
| | | | 2 - In addition to the above information, the report includes:<br>• The amounts of data transferred to and from each available coprocessor during the call. |
| `MIC_LD_LIBRARY_PATH` | None | Specifies the search path for coprocessor-side dynamic libraries. | Must contain the path to Intel MKL coprocessor-side libraries. The default path is `<mkl directory>/lib/mic`. |
| `MKL_MIC_THRESHOLDS_?GEMM` | None | Specifies matrix size thresholds for `?GEMM` computations in the AO mode. | Three comma-separated integers: *M*, *N*, *K*. If this environment variable is set, any call to a `?GEMM` function with problem sizes $M_1$, $N_1$, and $K_1$ tries to offload computations only if $M_1$>*M*, $N_1$>*N*, and $K_1$>*K*. This setting is only a hint, and Intel MKL may decide to not offload computations depending on the problem size and environment.<br>Example: To set the thresholds to *M*=2000, *N*=1000, *K*=500 for `DGEMM`, set `MKL_MIC_THRESHOLDS_DGEMM=2000,1000,500`. |

- Settings specified by the functions take precedence over the settings specified by the respective environment variables.
- Intel MKL interprets the values of `MKL_HOST_WORKDIVISION`, `MKL_MIC_WORKDIVISION`, and `MKL_MIC_<number>_WORKDIVISION` as guidance toward dividing work between coprocessors, but the library may choose a different work division if necessary.
- For LAPACK routines, setting the fraction of work to any value other than 0.0 enables the specified processor for AO mode. However Intel MKL LAPACK does not use the value specified to divide the workload. For example, setting the fraction to 0.5 has the same effect as setting the fraction to 1.0.

### See Also
Setting Environment Variables for Automatic Offload
Intel® Software Documentation Library for Intel® Compiler User and Reference Guides

## Setting Environment Variables for Automatic Offload

**Important**
To use Automatic Offload:

- If you completed the Scripts to Set Environment Variables Setting Environment Variables step of the Getting Started process, `MKL_MIC_ENABLE` is the only environment variable that you need to set.
- Otherwise, you must also set the `MIC_LD_LIBRARY_PATH` environment variable.

To set the environment variables for Automatic Offload mode, described in Automatic Offload Controls, use the appropriate commands:

`set MKL_MIC_ENABLE=1`

`set OFFLOAD_DEVICES=<list>`

For example: `set OFFLOAD_DEVICES=1,3`

`set OFFLOAD_ENABLE_ORSL=1`

`set MKL_HOST_WORKDIVISION=<value>`

For example: `set MKL_HOST_WORKDIVISION=0.2`

`set MKL_MIC_WORKDIVISION=<value>`

`set MKL_MIC_<number>_WORKDIVISION=<value>`

For example: `set MKL_MIC_2_WORKDIVISION=0.33`

`set MKL_MIC_MAX_MEMORY=<value>`

`set MKL_MIC_<number>_MAX_MEMORY=<value>`

For example: `set MKL_MIC_0_MAX_MEMORY=2G`

`set MKL_MIC_REGISTER_MEMORY=1`

`set MKL_MIC_RESOURCE_LIMIT=<value>`

For example: `set MKL_MIC_RESOURCE_LIMIT=0.34`

`set MIC_OMP_NUM_THREADS=<value>`

`set MIC_<number>_OMP_NUM_THREADS=<value>`

For example: `set MIC_0_OMP_NUM_THREADS=240`

`set OFFLOAD_REPORT=<level>`

For example: `set OFFLOAD_REPORT=2`

`set MIC_LD_LIBRARY_PATH="%MKLROOT%/lib/mic:%MIC_LD_LIBRARY_PATH%”`

`set MKL_MIC_THRESHOLDS_?GEMM="<N>,<M>,<K>"`

For example: `set MKL_MIC_THRESHOLDS_?GEMM="2000,1000,500"`

### See Also
Automatic Offload Controls
Detailed Directory Structure of the lib/intel64_win_mic Directory

# Compiler Assisted Offload

Compiler Assisted Offload is a method to offload computations to Intel Xeon Phi coprocessors that uses the Intel® compiler and its offload pragma support to manage the functions and data offloaded. See *Intel® Compiler User and Reference Guides* for more details.

> **Important**
> The Intel compilers support Intel MIC Architecture starting with version 13.

### See Also
Running Intel MKL on an Intel Xeon Phi Coprocessor in Native Mode

## Examples of Compiler Assisted Offload

The following are examples of Compiler Assisted Offload. Please see *Intel® Compiler User and Reference Guide* for more details.

These examples show how to call Intel MKL from offload regions that are executed on coprocessors based on Intel MIC Architecture and how to reuse data that already exists in the memory of the coprocessor and thus minimize data transfer.

### Fortran

### C

### See Also

Intel® Software Documentation Library for Intel® Compiler User and Reference Guides

## Linking for Compiler Assisted Offload

Intel MKL provides both static and dynamic libraries for coprocessors based on Intel MIC Architecture, but the Single Dynamic Library is unavailable for the coprocessors.

See Selecting Libraries to Link with for libraries to list on your link line in the simplest case.

See Detailed Directory Structure of the lib/intel64_win_intel64_lin_mic Directory for a full list of libraries provided in the `<mkl directory>\lib\intel64_win_mic` directory.

You can link either static or dynamic host-side libraries and either static or dynamic coprocessor-side libraries independently.

To run applications linked dynamically with the host-side and coprocessor-side libraries, perform the Scripts to Set Environment Variables Setting Environment Variables step of the Getting Started process. In addition to other environment variables, it sets:

- `path` to contain `<parent directory>\redist\intel64_win\mkl`
- `MIC_LD_LIBRARY_PATH` to contain `<mkl directory>\lib\intel64_win_mic`

> **Important**
> Because Intel MKL provides both LP64 and ILP64 interfaces, ensure that the host and coprocessor-side executables use the same interface or cast all 64-bit integers to 32-bit integers (or vice-versa) before calling coprocessor-side functions in your application.

The following examples illustrate linking for compiler assisted offload to Intel Xeon Phi coprocessors.

The examples use a `.c` (C) source file and Intel® C++ Compiler. Fortran users should instead specify a `.f` file and replace `icl` with `ifort`.

> **Important**
> Coprocessors run a Unix* operating system.

- Static linking of `myprog.f`, host-side and coprocessor-side libraries for parallel Intel MKL using LP64 interface:

  ```
  set lib=%MKLROOT%\lib\intel64_win;%lib%
  ```

  ```
  set include=%MKLROOT%\include;%include%
  ```

  ```
  icl /Qopenmp /MT /Qoffload-attribute-target=mic /Qoffload-option,mic,compiler,
  " -I%MKLROOT%\include" /Qoffload-option,mic,link," -Wl,--start-group
  ```

```
%MKLROOT%\lib\intel64_win_mic\libmkl_intel_lp64.a %MKLROOT%\lib\intel64_win_mic
\libmkl_core.a
%MKLROOT%\lib\intel64_win_mic\libmkl_intel_thread.a -Wl,--end-group"
myprog.c mkl_intel_lp64.lib mkl_intel_thread.lib mkl_core.lib
```

- Dynamic linking of `myprog.f`, host-side and coprocessor-side libraries for parallel Intel MKL using LP64 interface:

```
set lib=%MKLROOT%\lib\intel64_win;%lib%

set include=%MKLROOT%\include;%include%

icl /Qopenmp /MT /Qoffload-attribute-target=mic /Qoffload-option,mic,compiler,
" -I%MKLROOT%\include" /Qoffload-option,mic,link,"-L%MKLROOT%\lib\intel64_win_mic
-lmkl_intel_lp64 -lmkl_core -lmkl_intel_thread"
myprog.c mkl_intel_lp64_dll.lib mkl_intel_thread_dll.lib mkl_core_dll.lib
```

### See Also

Linking Your Application with the Intel® Math Kernel Library
Using the Link-line Advisor

# Using Automatic Offload and Compiler Assisted Offload in One Application

You can use Automatic Offload and Compiler Assisted Offload in the same application. However, to avoid oversubscription of computational resources of the coprocessors, synchronize Intel MKL and Intel Compiler accesses to coprocessors using either of the these techniques:

- In your code, manually synchronize #pragma offload regions and calls of Intel MKL functions that support Automatic Offload.
- Set the `OFFLOAD_ENABLE_ORSL` environment variable to 1 to enable automatic synchronization.

### See Also

Automatic Offload
Compiler Assisted Offload

# Running Intel MKL on an Intel Xeon Phi Coprocessor in Native Mode

Some applications can benefit from running on Intel Xeon Phi coprocessors in native mode. In this mode, the application runs directly on a coprocessor and its Linux* operating system without being offloaded from a host system. To run on Intel MIC Architecture in the native mode, an application requires minimal changes to the source code.

Because in the native mode the code runs exclusively on a coprocessor, binaries built for native runs contain only the code to be run on a coprocessor. Intel compilers provide a specialized option to support building applications to be run in the native mode.

To build an application that calls Intel MKL and natively run it on a coprocessor, you need to perform these high-level steps:

1. On the host system, compile and build the application using the `/Qmic` option.
2. Transfer the executable and all the dependencies it requires to the coprocessor:
   - The Intel MKL libraries in the `<mkl directory>\lib\intel64_win_mic` directory.
   - `libiomp5.so` in the `<parent directory>\compiler\lib\intel64_win_mic` directory.
3. Use the Telnet protocol to execute on the coprocessor and add the paths to the dynamic libraries transferred to the coprocessor in step 2 and to the value of the `LD_LIBRARY_PATH` environment variable.

4. Set the number of threads and the thread affinity using your threading run-time library.
5. Execute just as you would on a standard Linux* system.

For more information, see *Intel® Compiler User and Reference Guides*, available in the Intel Software Documentation Library.

### See Also

Detailed Directory Structure of the lib/intel64_win_mic Directory
Improving Performance on Intel Xeon Phi Coprocessors
Intel Software Documentation Library

## Using ScaLAPACK and Cluster FFT on Intel Xeon Phi Coprocessors

Intel MKL ScaLAPACK and Cluster FFT support only Intel MPI Library on Intel Xeon Phi coprocessors.

The Intel MPI library can treat each Intel Xeon Phi coprocessor as a regular node in a cluster of processors based on Intel 64 architecture and Intel Xeon Phi coprocessors and enables a straightforward way to run an MPI application on clusters that contain both processors and coprocessors as compute nodes.

The documentation for the Intel MPI library recommends the following steps to run an MPI application on the specific Intel Xeon Phi coprocessor and the host node if the nodes are properly specified on the cluster and the network protocols and environment are properly set up:

1. Build the application for the Intel 64 architecture.
2. Build the application for the Intel MIC Architecture.
3. Launch the application from the host computer.

---

**NOTE**
If you need to run the application on the coprocessor only, you can alternatively launch it from the coprocessor.

---

For more details, check the Intel MPI Library documentation, available in the Intel Software Documentation Library.

To run a dynamically linked application natively, perform the Scripts to Set Environment Variables Setting Environment Variables step of the Getting Started process. In addition to other environment variables, it sets:

- `path` to contain `<parent directory>\redist\intel64_win\mkl`
- `MIC_LD_LIBRARY_PATH` to contain `<mkl directory>\lib\intel64_win_mic`

When building your application that uses Intel MKL ScaLAPACK or Cluster FFT, follow the linking guidelines in the Linking with Intel MKL Cluster Software, but be aware of Intel MKL specifics on Intel Xeon Phi coprocessors:

- Only Intel compiler and Intel MPI are supported
- Only OpenMP threading layer for Intel compilers and Intel® Threading Building Blocks (Intel® TBB) threading layer are provided

You can find a full list of Intel MKL libraries for Intel MIC architecture in Detailed Directory Structure of the lib/intel64_win_intel64_lin_mic Directory. Be aware that coprocessors run a Unix* operating system.

---

**TIP**
Use the Using the Link-line Advisor to quickly choose the appropriate set of libraries and linker options.

---

### See Also

Linking Your Application with the Intel® Math Kernel Library
Intel Software Documentation Library

## Examples of Linking with ScaLAPACK and Cluster FFT for Intel(R) Many Integrated Core Architecture

### Examples of Linking a C Application

These examples illustrate linking of an application for Intel MIC Architecture under the following conditions:

- The application uses Intel MKL ScaLAPACK or Cluster FFT.
- Main module is in C.
- You are using the Intel® C++ Compiler.
- Your programming interface is LP64.
- *<path to mpi binaries>* is the path to Intel MPI binaries for Intel MIC Architecture.
- `$MKLPATH` is a user-defined variable that contains `$MKLROOT/lib/intel64_win_mic`

See the *Intel MKL Release Notes* for details of system requirements.

To link with ScaLAPACK for native runs on a cluster of systems based on the Intel MIC architecture, use the following link line:

```
<path to mpi binaries>/mpicc /Qmic <files to link> \
-L$MKLPATH \
-lmkl_scalapack_lp64 \
-lmkl_blacs_intelmpi_lp64 \
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core \
-liomp5 -lpthread -lm
```

To link with Cluster FFT for native runs on a cluster of systems based on the Intel MIC architecture, use the following link line:

```
<path to mpi binaries>/mpicc /Qmic <files to link> \
-Wl,--start-group \
$MKLPATH/libmkl_cdft_core.a \
$MKLPATH/libmkl_blacs_intelmpi_lp64.a \
$MKLPATH/libmkl_intel_lp64.a \
$MKLPATH/libmkl_intel_thread.a \
$MKLPATH/libmkl_core.a \
-Wl,--end-group \
-liomp5 -lpthread -lm
```

### See Also

Working with the Intel® Math Kernel Library Cluster Software
Using the Link-line Advisor

### Examples of Linking a Fortran Application

These examples illustrate linking of an application for Intel MIC Architecture under the following conditions:

- The application uses Intel MKL ScaLAPACK or Cluster FFT.
- Main module is in Fortran.
- You are using the Intel® Fortran Compiler.
- Your programming interface is LP64.
- *<path to mpi binaries>* is the path to Intel MPI binaries for Intel MIC Architecture.
- `$MKLPATH` is a user-defined variable that contains `$MKLROOT/lib/intel64_win_mic`.

See the *Intel MKL Release Notes* for details of system requirements.

To link with ScaLAPACK for native runs on a cluster of systems based on the Intel MIC architecture, use the following link line:

```
<path to mpi binaries>/mpiifort /Qmic <files to link> \
-L$MKLPATH \
```

```
-lmkl_scalapack_lp64 \
-lmkl_blacs_intelmpi_lp64 \
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core \
-liomp5 -lpthread -lm
```

To link with Cluster FFT for native runs on a cluster of systems based on the Intel MIC architecture, use the following link line:

```
<path to mpi binaries>/mpiifort /Qmic <files to link> \
-Wl,--start-group \
$MKLPATH/libmkl_cdft_core.a \
$MKLPATH/libmkl_blacs_intelmpi_lp64.a \
$MKLPATH/libmkl_intel_lp64.a \
$MKLPATH/libmkl_intel_thread.a \
$MKLPATH/libmkl_core.a \
-Wl,--end-group \
-liomp5 -lpthread -lm
```

### See Also

Working with the Intel® Math Kernel Library Cluster Software
Using the Link-line Advisor

# Threading Behavior of Intel MKL on Intel MIC Architecture

To avoid performance drops caused by oversubscribing Intel Xeon Phi coprocessors, Intel MKL limits the number of OpenMP threads it uses to parallelize computations:

- For native runs on coprocessors, Intel MKL uses 4**Number-of-Phi-Cores* threads by default and scales down the number of threads back to this value if you request more threads and `MKL_DYNAMIC` is true.
- For runs that offload computations, Intel MKL uses 4*(*Number-of-Phi-Cores*-1) threads by default and scales down the number of threads back to this value if you request more threads and `MKL_DYNAMIC` is true.
- If you request fewer threads than the default number, Intel MKL will use the requested number.

Here *Number-of-Phi-Cores* is the number of cores per coprocessor.

### See Also

MKL_DYNAMIC
Techniques to Set the Number of Threads

# Improving Performance on Intel Xeon Phi Coprocessors

To improve performance of Intel MKL on Intel Xeon Phi coprocessors, use the following tips, which are specific to Intel MIC Architecture. General performance improvement recommendations provided in Coding Techniques also apply.

For more information, see the Knowledge Base article at http://software.intel.com/en-us/articles/performance-tips-of-using-intel-mkl-on-intel-xeon-phi-coprocessor.

## Memory Allocation

Performance of many Intel MKL routines improves when input and output data reside in memory allocated with 2MB pages because this enables you to address more memory with less pages and thus reduce the overhead of translating between virtual and physical memory addresses compared to memory allocated with the default page size of 4K. For more information, refer to *Intel® 64 and IA-32 Architectures Optimization Reference Manual* and *Intel® 64 and IA-32 Architectures Software Developer's Manual* (connect to http://www.intel.com/ and enter the name of each document in the **Find Content** text box).

To allocate memory with 2MB pages, you can use the `mmap` system call with the `MAP_HUGETLB` flag. You can alternatively use the `libhugetlbfs` library. See the white paper at http://software.intel.com/sites/default/files/Large_pages_mic_0.pdf for more information.

To enable allocation of memory with 2MB pages for data of size exceeding 2MB and transferred with offload pragmas, set the `MIC_USE_2MB_BUFFERS` environment variable to an appropriate value. This setting ensures that all pointer-based variables whose run-time length exceeds this value will be allocated in 2MB pages. For example, with `MIC_USE_2MB_BUFFERS=64K`, variables with run-time length exceeding 64 KB will be allocated in 2MB pages. For more details, see *Intel® Compiler User and Reference Guides*, available in the Intel Software Documentation Library.

Specifying the maximum amount of memory on a coprocessor that can be used for Automatic Offload computations typically enhances the performance by enabling Intel MKL to reserve and keep the memory on the coprocessor during Automatic Offload computations. You can specify the maximum memory by setting the `MKL_MIC_MAX_MEMORY` environment variable to a value such as 2 GB.

## Data Alignment and Leading Dimensions

To improve performance of Intel MKL FFT functions, follow these recommendations:

- Align the first element of the input data on 64-byte boundaries
- For two- or higher-dimensional single-precision transforms, use leading dimensions (strides) divisible by 8 but not divisible by 16
- For two- or higher-dimensional double-precision transforms, use leading dimensions divisible by 4 but not divisible by 8

For other Intel MKL function domains, use general recommendations for data alignment.

## Number of Threads

For FFT, use a number of threads depending on the total size of the input and output data for the transform:

- A power of two, if the total size is less than *Number-of-Phi-Cores**0.5 MB
- 4**Number-of-Phi-Cores,* if the total size is greater than *Number-of-Phi-Cores**0.5 MB

Here *Number-of-Phi-Cores* is the number of Intel Xeon Phi coprocessors on the system.

For more information, see Improving Performance with Threading and SettingDetermining the Number of OpenMP* Threads.

## OpenMP Thread Affinity

To improve performance of Intel MKL routines, set `KMP_AFFINITY=balanced` for all function domains.

## Intel® Threading Building Blocks Facilities

To improve performance of Intel MKL routines, use the `tbb::affinity_partitioner` class.

To adjust the number of threads (for example, see Number of Threads for FFT), use the `tbb::task_scheduler_init` class.

For more information, see the Intel® TBB documentation at https://www.threadingbuildingblocks.org/documentation.

## See Also

Examples of Compiler Assisted Offload
Intel Software Documentation Library

# Managing Behavior of the Intel(R) Math Kernel Library with Environment Variables

11

## Managing Behavior of Function Domains

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

Intel® Math Kernel Library (Intel® MKL) 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 MKL Reference Manual* and the description of the `vmlSetMode` function in particular.

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

```
set MKL_VML_MODE=<mode-string>
```

In this command, *<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 |
| `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. |
| Error Mode Control | |
| `VML_ERRMODE_IGNORE` | No action is set for computation errors. |
| `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. |

| Value of *mode* | Description |
|---|---|
| VML_ERRMODE_DEFAULT | On error, an exception is raised and an additional error handler function is called. |

This command provides an example of valid settings for the MKL_VML_MODE environment variable:

```
set 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 MKL 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 MKL 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>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>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 MKL Reference Manual* for details. |
| | 1 | Configures CFFT to merge local FFT calls with local transpositions.<br>CFFT usually performs faster with this value than with wo_omatcopy = 0 if DFTI_TRANSPOSE has the value of DFTI_NONE. |

| Option | Possible Values | Description |
|---|---|---|
| | -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:

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

**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

# Enabling Intel® MKL to Dispatch New Intel® Architectures

Intel® MKL provides support for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instruction-set architecture (ISA) ahead of hardware availability, but the CPU does not automatically dispatch this ISA by default. To turn on automatic CPU-based dispatching for systems based on Intel® Xeon® processors that support this ISA (or in an appropriate simulated environment), do either of the following:

- Call

  `mkl_enable_instructions(MKL_ENABLE_AVX512)`
- Set the environment variable:

  `set MKL_ENABLE_INSTRUCTIONS=AVX512`

For more details of the `mkl_enable_instructions` function, see the *Intel MKL Reference Manual*.

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

# Programming with Intel® Math Kernel Library in Integrated Development Environments (IDE)

12

## Configuring Your Integrated Development Environment to Link with Intel(R) Math Kernel Library

See these Knowledge Base articles for how to configure your Integrated Development Environment for linking with Intel MKL:

- Compiling and Linking Intel® MKL with Microsoft* Visual C/C++* (http://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-compiling-and-linking-with-microsoft-visual-cc)
- How to Build an Intel® MKL Application with Intel® Visual Fortran Compiler (http://software.intel.com/en-us/articles/how-to-build-mkl-application-in-intel-visual-fotran-msvc2005)
- Configuring Intel® MKL in Microsoft* Visual Studio* (http://software.intel.com/en-us/articles/configuring-intel-mklin-microsoft-visual-studio)

### Configuring the Microsoft Visual C/C++* Development System to Link with Intel® MKL

Steps for configuring Microsoft Visual C/C++* development system for linking with Intel® Math Kernel Library (Intel® MKL) depend on whether you installed the C++ Integration(s) in Microsoft Visual Studio* component of the Intel® Parallel Studio XE Composer Edition:

- If you installed the integration component, see Automatically Linking Your Microsoft Visual C/C++* Project with Intel® MKL.
- If you did not install the integration component or need more control over Intel MKL libraries to link, you can configure the Microsoft Visual C++* development system by performing the following steps. Though some versions of the Visual C++* development system may vary slightly in the menu items mentioned below, the fundamental configuring steps are applicable to all these versions.

  1. In Solution Explorer, right-click your project and click **Properties**
  2. Select **Configuration Properties** > **VC++ Directories**
  3. Select **Include Directories**. Add the directory for the Intel MKL include files, that is, *<mkl directory>*\include
  4. Select **Library Directories**. Add architecture-specific directories for Intel MKL and OpenMP* libraries, for example: *<mkl directory>*\lib\ia32_win and *<parent directory>*\compiler\lib\ia32_win
  5. Select **Executable Directories**. Add architecture-specific directories with dynamic-link libraries:
     - For OpenMP* support, for example: *<parent directory>*\redist\ia32_win\compiler
     - For Intel MKL (only if you link dynamically), for example: *<parent directory>*\redist\ia32_win\mkl
  6. Select **Configuration Properties** > **Custom Build Setup** > **Additional Dependencies**. Add the libraries required, for example, mkl_intel_c.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib

#### See Also

Intel® Software Documentation Library for the documentation for Intel Parallel Studio XE Composer Edition

Linking in Detail
Notational Conventions

### Configuring Intel® Visual Fortran to Link with Intel MKL

Steps for configuring Intel® Visual Fortran for linking with Intel® Math Kernel Library (Intel® MKL) depend on whether you installed the Visual Fortran Integration(s) in Microsoft Visual Studio* component of the Intel® Parallel Studio XE Composer Edition:

- If you installed the integration component, see Automatically Linking Your Intel® Visual Fortran Project with Intel® MKL.
- If you did not install the integration component or need more control over Intel MKL libraries to link, you can configure your project as follows:

  1. Select **Project** > **Properties** > **Linker** > **General** > **Additional Library Directories**. Add architecture-specific directories for Intel MKL and OpenMP* libraries, for example: `<mkl directory>\lib\ia32_win` and `<parent directory>\compiler\lib\ia32_win`
  2. Select **Project** > **Properties** > **Linker** > **Input** > **Additional Dependencies**. Insert names of the required libraries, for example: `mkl_intel_c.lib mkl_intel_thread.lib mkl_core.lib libiomp5md.lib`
  3. Select **Project** > **Properties** > **Debugging** > **Environment**. Add architecture-specific paths to dynamic-link libraries:

     - For OpenMP* support; for example: enter `PATH=%PATH%;<parent directory>\redist\ia32_win\compiler`
     - For Intel MKL (only if you link dynamically); for example: enter `PATH=%PATH%;<parent directory>\redist\ia32_win\mkl`

#### See Also

Intel® Software Documentation Library for the documentation for Intel Parallel Studio XE Composer Edition
Notational Conventions

# Getting Assistance for Programming in the Microsoft Visual Studio* IDE

### Using Context-Sensitive Help

You can get context-sensitive help when typing your code in the Visual Studio* IDE Code Editor. To open the help topic describing an Intel MKL function called in your code, select the function name and press F1. The topic with the function description opens in the Microsoft Help Viewer or your Web browser depending on the Visual Studio IDE Help settings.

### Using the IntelliSense* Capability

IntelliSense is a set of native Visual Studio*(VS) IDE features that make language references easily accessible.

The user programming with Intel MKL in the VS Code Editor can employ two IntelliSense features: Parameter Info and Complete Word.

Both features use header files. Therefore, to benefit from IntelliSense, make sure the path to the include files is specified in the VS or solution settings. For example, see Configuring the Microsoft Visual C/C++* Development System to Link with Intel® MKL on how to do this.

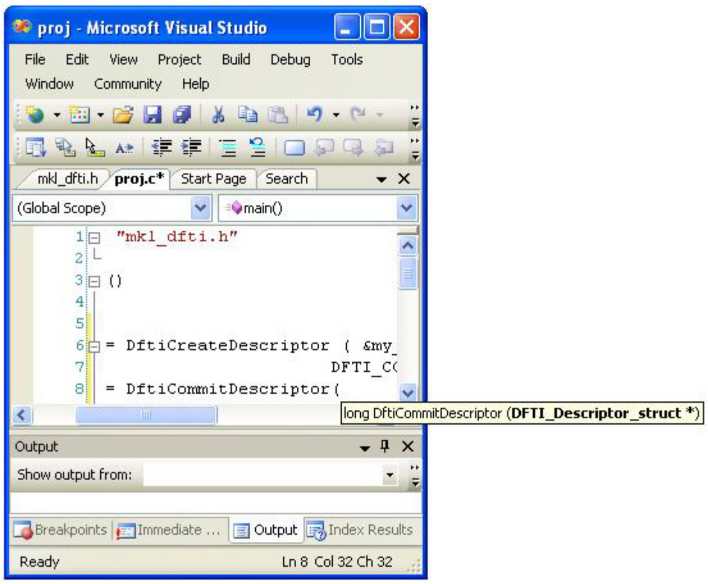
## Parameter Info

The Parameter Info feature displays the parameter list for a function to give information on the number and types of parameters. This feature requires adding the `include` statement with the appropriate Intel MKL header file to your code.

To get the list of parameters of a function specified in the header file,

1. Type the function name.
2. Type the opening parenthesis.

This brings up the tooltip with the list of the function parameters:

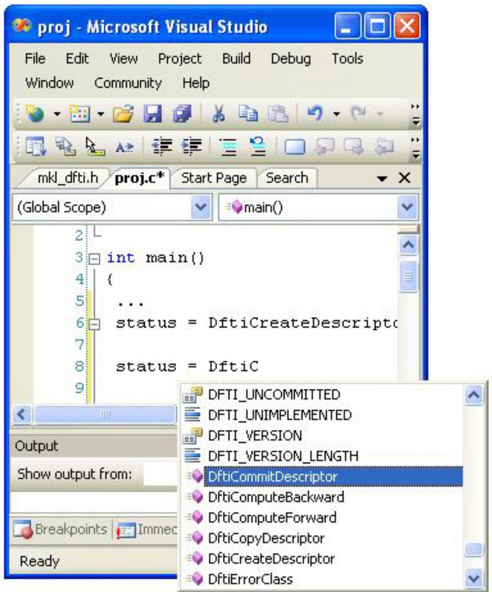



## Complete Word

For a software library, the Complete Word feature types or prompts for the rest of the name defined in the header file once you type the first few characters of the name in your code. This feature requires adding the `include` statement with the appropriate Intel MKL header file to your code.

To complete the name of the function or named constant specified in the header file,

1. Type the first few characters of the name.
2. Press Alt+RIGHT ARROW or Ctrl+SPACEBAR.
   If you have typed enough characters to disambiguate the name, the rest of the name is typed automatically. Otherwise, a pop-up list appears with the names specified in the header file
3. Select the name from the list, if needed.


proj - Microsoft Visual Studio
File Edit View Project Build Debug Tools
Window Community Help
mkl_dfti.h proj.c* Start Page Search
(Global Scope)
main()
2
3 int main()
4 {
5 ...
6 status = DftiCreateDescripto
7
8 status = DftiC
9
Output
Show output from:
Breakpoints Immec
Ready
DFTI_UNCOMMITTED
DFTI_UNIMPLEMENTED
DFTI_VERSION
DFTI_VERSION_LENGTH
DftiCommitDescriptor
DftiComputeBackward
DftiComputeForward
DftiCopyDescriptor
DftiCreateDescriptor
DftiErrorClass

# LINPACK and MP LINPACK Benchmarks

13

## Intel® Optimized LINPACK Benchmark for Windows* OS

Intel® Optimized LINPACK Benchmark 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:

- MP LINPACK, which is a distributed memory version of the same benchmark.
- 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 http://www.intel.com/software/products/.

### 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 Windows* OS contains the following files, located in the `benchmarks\linpack\` subdirectory of the Intel® Math Kernel Library (Intel® MKL) directory:

| File in `benchmarks\linpack\` | Description |
|---|---|
| `linpack_xeon32.exe` | The 32-bit program executable for a system based on Intel® Xeon® processor or Intel® Xeon® processor MP with or without Streaming SIMD Extensions 3 (SSE3). |
| `linpack_xeon64.exe` | The 64-bit program executable for a system with Intel Xeon processor using Intel® 64 architecture. This program may accelerate execution by using Intel® Xeon Phi™ coprocessors if they are available on the system. |
| `xlinpack_mic` | The 64-bit program executable for a native run on an Intel Xeon Phi coprocessor. |
| `runme_xeon32.bat` | A sample shell script for executing a pre-determined problem set for `linpack_xeon32.exe`. |
| `runme_xeon64.bat` | A sample shell script for executing a pre-determined problem set for `linpack_xeon64.exe`. |

| File in benchmarks \linpack\ | Description |
|---|---|
| runme_xeon64_ao.bat | A sample shell script for executing a pre-determined problem set for linpack_xeon64.exe. The script enables acceleration by offloading computations to Intel Xeon Phi coprocessors available on the system. |
| runme_mic | A sample shell script for executing a pre-determined problem set for xlinpack_mic. |
| lininput_xeon32 | Input file for a pre-determined problem for the runme_xeon32 script. |
| lininput_xeon64 | Input file for a pre-determined problem for the runme_xeon64 script. |
| lininput_xeon64_ao | Input file for a pre-determined problem for the runme_xeon64_ao script. |
| lininput_mic | Input file for a pre-determined problem for the runme_mic script. |
| 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. | |
| win_xeon32.txt | Result of the runme_xeon32 script execution. |
| win_xeon64.txt | Result of the runme_xeon64 script execution. |
| win_xeon64_ao.txt | Result of the runme_xeon64_ao script execution. |
| win_mic.txt | Result of the runme_mic 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 one of the following, as appropriate:

```
runme_xeon32.bat
runme_xeon64.bat
runme_xeon64_ao.bat
runme_mic
```

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

```
linpack_xeon32.exe -e
linpack_xeon64.exe -e
./xlinpack_mic -e
```

The pre-defined data input fileslininput_xeon32 and lininput_xeon64 are examples. Different systems have different numbers of processors or amounts of memory and therefore require new input files. The extended help can give insight into proper ways to change the sample input files.

Each input file requires at least the following amount of memory:

| | |
|---|---|
| lininput_xeon32 | 2 GB |
| lininput_xeon64 | 16 GB |

| | |
|---|---|
| `lininput_xeon64_ao` | 8 GB |
| `lininput_mic` | 8 GB |

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.

| Optimization Notice |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. |
| Notice revision #20110804 |

## Known Limitations of the Intel® Optimized LINPACK Benchmark

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

- Intel Optimized LINPACK Benchmark supports only OpenMP threading
- Intel Optimized LINPACK Benchmark is threaded to effectively use multiple processors. So, in multi-processor systems, best performance will be obtained with the Intel® Hyper-Threading Technology turned off, which ensures that the operating system assigns threads to physical processors only.
- 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.

# Intel® Optimized MP LINPACK Benchmark for Clusters

## Overview of the Intel Optimized MP LINPACK Benchmark

The Intel® Optimized MP LINPACK Benchmark for Clusters (Intel® Optimized MP LINPACK Benchmark) is based on modifications and additions to High-Performance LINPACK (HPL) 2.1 (http://www.netlib.org/benchmark/hpl) from Innovative Computing Laboratories (ICL) at the University of Tennessee, Knoxville. The Intel Optimized MP LINPACK Benchmark can be used for TOP500 runs (see http://www.top500.org). To use the benchmark you need be familiar with HPL usage. The Intel Optimized MP LINPACK Benchmark provides some additional enhancements designed to make the HPL usage more convenient and to use Intel® Message-Passing Interface (MPI) settings that may enhance performance. The `.\benchmarks\mp_linpack` directory adds techniques to minimize search times frequently associated with long runs.

The Intel® Optimized MP LINPACK Benchmark implements the Massively Parallel (MP) LINPACK benchmark using HPL code. It solves a random dense system of linear equations ($Ax=b$) in `real*8` precision, 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. You are not limited to solving a number of equations $N$ equal to 1000 because the implementation can be generalized to solve any size system of equations that meets the restrictions imposed by the MPI implementation chosen. The benchmark uses proper random number generation technique and full row pivoting to ensure the accuracy of the results.

Do not use this benchmark to report LINPACK 100 performance. Do not confuse this benchmark with:

- LINPACK, the library, which has been expanded upon by the LAPACK library.
- Intel Optimized LINPACK Benchmark, which is a shared memory (SMP) version of the same benchmark. While the Intel Optimized MP LINPACK Benchmark can be run on both a single node and a cluster, the Intel Optimized LINPACK Benchmark can only be run on a single node.

Intel provides optimized versions of the LINPACK benchmarks to help you obtain high LINPACK benchmark results on your systems based on genuine Intel processors more easily than with the standard HPL benchmark. Use the Intel Optimized MP LINPACK Benchmark to benchmark your cluster. The prebuilt binaries require Intel® MPI library be installed on the cluster.

> **NOTE**
> Intel Optimized MP LINPACK Benchmark prebuilt binaries cannot run with the symmetric model of the Intel MPI library. For details, see the article at https://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems.

The run-time version of Intel MPI library is free and can be downloaded from http://www.intel.com/software/products/ .

> **NOTE**
> The prebuilt binaries are hybrid offload binaries. They work even when the system does not have any Intel® Xeon Phi™ coprocessors.

The Intel package includes software developed at the University of Tennessee, Knoxville, ICL, and neither the University nor ICL endorse or promote this product. Although HPL 2.1 is redistributable under certain conditions, this particular package is subject to the Intel MKL license.

Intel MKL has introduced a new *hybrid* build functionality into Intel Optimized MP LINPACK Benchmark, while continuing to support the previous, non-hybrid build. The term *hybrid* refers to special optimizations added to take advantage of mixed OpenMP*/MPI parallelism.

If you want to use one MPI process per node and to achieve further parallelism by means of OpenMP, use the hybrid build. In general, the hybrid build is useful when the number of MPI processes per core is less than one. If you want to rely exclusively on MPI for cross-node parallelism and use one MPI process per core, use the non-hybrid build.

In addition to supplying certain hybrid prebuilt binaries, Intel MKL supplies some hybrid prebuilt libraries for Intel® MPI to take advantage of the additional OpenMP optimizations.

To enable you to offload computations from recent Intel® Xeon® processors to between zero and eight Intel Xeon Phi coprocessors, Intel MKL supplies a hybrid offload binary. The hybrid offload binary contains the latest optimizations for Intel® Core™ processors, and you are encouraged to use this binary even when the system does not have any Intel Xeon Phi coprocessors. The hybrid offload binary uses system-specific threading APIs to exploit mixed parallelism.

If you want to use an MPI version other than Intel MPI, you can do so by using the MP LINPACK source code provided. You can use the source code to build a non-hybrid version that may be used in a hybrid mode, but it would be missing some of the optimizations added to the hybrid version.

Non-hybrid builds are the default of the source code makefiles provided. To use the non-hybrid code in a hybrid mode, use the threaded version of Intel MKL BLAS, link with a thread-safe MPI (for example: use the `–mt_mpi` option with Intel MPI library), and call function `MPI_init_thread()` so that MPI is thread-safe.

| Optimization Notice |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. |

| **Optimization Notice** |
| --- |
| Notice revision #20110804 |

## Usage Modes of Intel Optimized MP LINPACK Benchmark for Intel® Xeon Phi™ Coprocessors

Intel MKL supports Intel® Xeon Phi™ coprocessors in the Hybrid Offload mode.

The Hybrid Offload mode combines use of different parallelization methods and offloading computations to coprocessors. In this mode, the host processor uses fewer cores for MPI than the total number of physical cores, also uses OpenMP* or Windows* OS threads, and offloads chunks of the problem to the Intel Xeon Phi coprocessor.

In many cases, the host Intel Xeon processor has more memory than the Intel Xeon Phi coprocessor. Therefore, the MPI processes have access to more memory when run on the host processors than on the coprocessors.

HPL code is homogeneous by nature: it requires that each MPI process runs in an environment with similar CPU and memory constraints. If for some reason, one node is twice as powerful as another node, in the past you could balance this only by running two MPI processes on the faster node.

Intel MKL now supports heterogeneous Intel Optimized MP LINPACK Benchmark. Heterogeneity means that Intel MKL supports a data distribution that can be balanced to the performance requirements of each node if there is enough memory on that node to support any additional work. The Intel Optimized MP LINPACK Benchmark supports:

- Intra-node heterogeneity,

  where a node includes different processing units with different compute capabilities. To use intra-node heterogeneity, where work is shared between the Intel Xeon processors and Intel Xeon Phi coprocessors, use the hybrid offload techniques.
- Inter-node heterogeneity,

  where the nodes themselves can differ. For information on how to configure Intel MKL to use the inter-node heterogeneity, see Heterogeneous Intel Optimized MP LINPACK Benchmark.

To maximize performance, increase the memory on the host processor or processors (64 GB per coprocessor is ideal) and run a large problem and large block-size. Such runs offload pieces of work to the coprocessors. Although this method increases the PCIe bus traffic, it is worthwhile for solving a problem that is large enough.

## Contents of the Intel Optimized MP LINPACK Benchmark

The Intel Optimized MP LINPACK Benchmark includes the entire HPL 2.1 distribution, but with modifications to several files, and files specific to the Intel Optimized MP LINPACK Benchmark. All the files are located in the `benchmarks\mp_linpack\` subdirectory of the Intel MKL directory. Files specific to the Intel Optimized MP LINPACK Benchmark are listed below.

**NOTE**
Because the Intel Optimized MP LINPACK Benchmark includes the *entire* HPL 2.1 distribution, which provides a configuration for Linux* OS only, some Linux OS files remain in the directory.

| **Directory/File in** *<mkl directory>* `\benchmarks\mp_linpack\` | **Contents** |
| --- | --- |
| `Make` | Sample architecture makefile for nmake utility to be used on processors based on the IA-32 and Intel® 64 architectures. |

| Directory/File in *<mkl directory>* \benchmarks\mp_linpack\ | Contents |
|---|---|
| HPL.dat | A duplicate of testing\ptest\HPL.dat in the top-level directory. |
| Prebuilt executables for simple performance testing | |
| bin_intel \intel64\xhpl_intel64.exe | Prebuilt binary for the Intel® 64 architecture and Intel MPI library‡. The binary accelerates execution by offloading computations to Intel Xeon Phi coprocessors if they are available on the system. |
| Prebuilt libraries | |
| lib_hybrid \intel64\libhpl_hybrid.lib | Prebuilt library with the hybrid version of the Intel Optimized MP LINPACK Benchmark for the Intel® 64 architecture and Intel MPI library. |
| lib_hybrid \intel64\libhpl_offload.lib | Prebuilt library with the offload version of the Intel Optimized MP LINPACK Benchmark for the Intel® 64 architecture and Intel MPI library. |
| Run scripts and examples of input files | |
| bin_intel \intel64\runme_intel64.bat | Sample run script for the Intel® 64 architecture and hybrid offload binary. |
| bin_intel\intel64\runme_intel64_prv.bat | A script that sets HPL environment variables based on the number of Intel Xeon Phi coprocessors per MPI process. The script is called by runme_intel64.bat. |
| bin_intel\intel64\HPL.dat | Example of an input file for the Intel® 64 architecture and offload binary. |
| nodeperf.c | Sample utility that tests the DGEMM speed across the cluster. |

‡ For a list of supported versions of the Intel MPI Library, see system requirements in the Intel MKL Release Notes.

### See Also

High-level Directory Structure

## Compile Options Specific to the Intel Optimized MP LINPACK Benchmark

The Intel Optimized MP LINPACK Benchmark has some additional compile options available over the standard HPL 2.1 distribution. These new options are:

**ASYOUGO:** Provides non-impacting performance information while runs proceed. There are only a few outputs and this information does not impact performance. This can be useful for runs that otherwise can execute for hours without providing any information.

**ASYOUGO2:** Provides additional performance information at a modest performance cost by intercepting every DGEMM call.

**ASYOUGO2_DISPLAY:** Displays the performance of all the significant DGEMM calls in the run.

**ENDEARLY:** Displays performance hints and then terminates the run early.

**FASTSWAP:** Inserts the LAPACK-optimized DLASWP into the HPL code. You can experiment with this to determine best results.

**HYBRID:** Establishes the Hybrid OpenMP/MPI mode of MP LINPACK, enabling the use of threaded Intel MKL and prebuilt MP LINPACK hybrid libraries.

**CAUTION**
Use this option only with an Intel® compiler and the Intel® MPI library version 3.1 or higher.

## Building the Intel Optimized MP LINPACK Benchmark

The Intel Optimized MP LINPACK Benchmark contains a few sample architecture makefiles, which you can edit to fit your configuration. Specifically:

- Set `TOPdir` to the directory where the Intel Optimized MP LINPACK Benchmark is being built.
- Set MPI variables: `MPdir`, `MPinc`, and `MPlib`.
- Specify the location of Intel MKL and of files to be used (`LAdir`, `LAinc`, `LAlib`).
- Adjust compiler and compiler/linker options.
- Specify the architecture by setting the architecture parameter `arch` for the `make` command:
  - `ia32` - for MPI processes to run on a 32-bit host processor.
  - `intel64` - for MPI processes to run on a 64-bit host processor and use the Intel MKL hybrid offload methodology if Intel Xeon Phi coprocessors are available on the cluster.
- By default, the non-hybrid version of MP LINPACK is built. To build a different version, set the `version` parameter for the `nmake` command to `hybrid` or `offload`, as appropriate.

  For example:

  ```
  nmake arch=intel64 mpi=intelmpi version=offload install
  ```

The makefile creates the binary in the `bin\`*`<arch>`* subdirectory of your `mp_linpack` directory, where *`<arch>`* is the value of the architecture parameter in the `nmake` command (for example, `bin\intel64`).

The makefiles contain common values for some sample cases. To change these values, you must be familiar with building an HPL and picking appropriate values for the variables in the makefiles.

## Ease-of-use Command-line Parameters

The Intel Optimized MP LINPACK Benchmark supports command-line parameters for HPL that help you to avoid making small changes in the `HPL.dat` input files every time you do a new run.

Placeholders in this command line illustrate these parameters:

```
xhpl -n <problem size> -m <memory size in Mbytes> -b <block size> -p <grid row dimn> -q <grid column dimn>
```

You can also use command-line parameters to run the prebuilt binary and runme script. For example:

```
xhpl_intel64 -n <problem size> -m <memory size in Mbytes> -b <block size> -p <grid row dimn> -q <grid column dimn>
```

```
runme_intel64 -n <problem size> -m <memory size in Mbytes> -b <block size> -p <grid row dimn> -q <grid column dimn>
```

For more command-line parameters, see Heterogeneous Intel Optimized MP LINPACK Benchmark.

If you want to run for *N*=10000 on a 1x3 grid, execute this command, provided that the other parameters in `HPL.dat` are correct:

```
xhpl -n 10000 -p 1 -q 3
```

By using the `m` parameter you can scale by the memory size instead of the problem size. The `m` parameter only refers to the size of the matrix storage and not to the coprocessor memory size or other buffers. So if you want to use matrices that fit in 50000 Mbytes with *NB*=1024 on 16 nodes, 32 nodes, and 128 nodes, execute these commands:

```
xhpl -m 50000 -b 1024 -p 4 -q 4
```

```
xhpl -m 50000 -b 1024 -p 4 -q 8
```

```
xhpl -m 50000 -b 1024 -p 8 -q 16
```

## Heterogeneous Intel Optimized MP LINPACK Benchmark

The Intel Optimized MP LINPACK Benchmark supports both inter-node and intra-node heterogeneity.

Intel MKL achieves heterogeneous support by distributing the matrix data unequally between the nodes. You do not need to run a different number of MPI processes on the faster nodes. Heterogeneous support can work by running only one MPI process per node (although you can run more MPI processes) and assigning more work to the more powerful nodes. The heterogeneous factor command-line parameter `f` controls the amount of work to be assigned to the more powerful nodes:

```
xhpl -n 100000 -b 1024 -p 8 -q 16 -f <heterogeneous factor>
```

The heterogeneous factor does not need to be integer. If the heterogeneous factor is 2.5, roughly 2.5 times the work will be put on the more powerful nodes. The heterogeneous factor is an important tuning parameter. The more work you put on the more powerful nodes, the more memory you might be wasting on the other nodes. Intel MKL achieves load balance by distributing the work unevenly. You can balance for speed or balance for memory, but not both. If your cluster includes many different types of nodes, you may need multiple heterogeneous factors.

Suppose you have a cluster with 32 GB per node, but some of the nodes are twice as fast, so you put twice as much work on them. Because they only have 32 GB per node as well, all the other nodes must be treated as 16 GB nodes, and effectively waste half the memory. Note that wasting memory might require a smaller problem size, which lowers the overall performance.

Let *P* be the number of rows and *Q* the number of columns in your processor grid (*PxQ*). The work must be *homogeneous* within each processor column because vertical operations, such as pivoting, are synchronizing operations.

When there are two different types of nodes, use MPI to process all the faster nodes first, and make sure the "`PMAP process mapping`" (line 9) of `HPL.dat` is set to 1, for `Column-major` mapping. Because all the nodes must be the same within a process column, the number of faster nodes must always be a multiple of *P*, and you can specify the faster nodes by setting the number of process columns *C* for the faster nodes with the `c` command-line parameter:

```
xhpl -n 100000 -b 1024 -p 8 -q 16 -f <heterogeneous factor> -c <number of faster
processor columns>
```

Use both `f` and `c` command-line parameters together. The `-f 1.0 -c 0` setting returns you to the default homogeneous behavior.

To understand how to choose the problem size *N* for a heterogeneous run, first consider a homogeneous system, where you might choose *N* as follows:

*N ~= sqrt(Memory Utilization * P * Q * Memory Size in GBytes / 8)*

*Memory Utilization* is usually around 0.8 for homogeneous Intel Xeon processor systems. With Intel Xeon Phi coprocessors involved, *Memory Utilization* is probably around 0.7 due to extra buffers needed for communication.

On a heterogeneous system, you might apply a different formula for *N* for each "cluster" of nodes that are the same and take the minimum *N* involved. Suppose you have a cluster with only one heterogeneous factor *F* and the number of processor columns (out of the total *Q*) in the group with that heterogeneous factor equal to *C*. That group contains *P*C* nodes. First compute the sum of the parts: *S =F*P*C + P*(Q-C)*. Note that *S=P*Q,F*=1, and *C=Q* on a homogeneous system. So take *N* as

*N ~= sqrt(Memory Utilization * P * Q * ((F*P*C)/S) * Memory Size in GBytes / 8)*

or simply scale down the value of *N* for the homogeneous system by *sqrt(F*P*C/S)*.

### Example

Suppose the cluster has 100 nodes each having 64 GB of memory, and 20 of the nodes are 2.7 times as powerful as the other 80. Run *one* MPI process per node for a total of 100 MPI processes. Assume a square processor grid *P=Q*=10, which conveniently divides up the faster nodes evenly. Normally, the HPL documentation recommends choosing a matrix size that consumes 80 percent of available memory. If *N* is the size of the matrix, the matrix consumes 8*N*^2/(*P*Q*). So a homogeneous run might look like:

```
xhpl -n 820000 -b 256 -p 10 -q 10
```

Unfortunately, the HPL from Netlib runs this problem as slow as the slowest node, and all the extra performance potential from the faster nodes is lost. However, if you redistribute the matrix and run the heterogeneous Intel Optimized MP LINPACK Benchmark, you can take advantage of the faster nodes. But because some of the nodes will contain 2.7 times as much data as the other nodes, you must shrink the problem size (unless the faster nodes also happen to have 2.7 times as much memory). Instead of 0.8*64GB*100 total memory size, we have only 0.8*64GB*20 + 0.8*64GB/2.7*80 total memory, which is less than half the original space. So the problem size in this case would be 526000. If the faster nodes are faster because of the presence of Intel Xeon Phi coprocessors (intra-node heterogeneity), you might need to choose a larger block size as well (which reduces scalability to some extent). Because *P*=10 and there are 20 faster nodes, two processor columns are faster. If you arrange MPI to send these nodes first to the application, the command line looks like:

```
xhpl -n 526000 -b 1024 -p 10 -q 10 -f 2.7 -c 2
```

The `m` parameter may be misleading for heterogeneous calculations, because it calculates the problem size assuming all the nodes have the same amount of data.

**WARNING**
The number of faster nodes must be *C***P*. If the number of faster nodes is not divisible by *P*, you might not be able to take advantage of the extra performance potential by giving the faster nodes extra work.

While it suffices to provide `f` and `c` command-line parameters if you need one heterogeneous factor, to support multiple heterogeneous factors, you must add lines to the `HPL.dat` input as explained below.

For example, if there are three different types of nodes in a cluster and you need at least two heterogeneous factors:

1. Add these lines to the end of the `HPL.dat`:

```
1          number of heterogeneous factors
0 3 2.7   [start_column, stop_column, heterogeneous factor for that range]
```

**NOTE**
Numbering of processor columns starts at 0. The start and stopping numbers must be between 0 and *Q*-1 (inclusive).

You can also perform this step instead of providing the command-line parameters if you need only one heterogeneous factor.

2. For two heterogeneous factors, change the number in the first row above from 1 to 2 and follow that line with two lines specifying the start column, stopping column, and heterogeneous factor.

When choosing parameters for heterogeneous support in `HPL.dat`, primarily focus on the most powerful nodes. However, the larger the heterogeneous factor, the more balanced the cluster may be from a performance viewpoint, but the more imbalanced from a memory viewpoint. At some point, further performance balancing might affect the memory too much. If this is the case, try to reduce any changes done for the faster nodes (such as in block sizes). Experiment with values in `HPL.dat` carefully because wrong values may greatly hinder performance.

When tuning on a heterogeneous cluster, do not immediately attempt a heterogeneous run, but do the following:

1. Break the cluster down into multiple homogeneous clusters.
2. Make heterogeneous adjustments for performance balancing. For instance, if you have two different sets of nodes where one is three times as powerful as the other, it must do three times the work.
3. Figure out the approximate size of the problem (per node) that you can run on each piece.
4. Do some homogeneous runs with those problem sizes per node and the final block size needed for the heterogeneous run and find the best parameters.

5. Use these parameters for an initial heterogeneous run.

## Running the Intel Optimized MP LINPACK Benchmark

### See Also

Building the Intel Optimized MP LINPACK Benchmark

### Running the Benchmark on One Node

To run the Intel Optimized MP LINPACK Benchmark binary on a cluster node, follow the steps below.

**NOTE**
While these instructions assume the Intel® 64 architecture, they are more widely applicable. The instructions directly apply to Previous Generation Intel® Core™ or higher Intel® processors. For IA-32 architecture processors and for earlier Intel® 64 architecture processors, omit the `version` parameter of the `make` command. For IA-32 architecture processors, also adjust directory names and the value of the `arch` parameter.

1. Load the necessary environment variables for Intel MKL, Intel MPI, and the Intel® compiler and build the binary:

   ```
   <parent directory>\bin\compilervars.bat intel64
   ```

   ```
   <mpi directory>\bin64\mpivars.bat
   ```

   ```
   <mkl directory>\bin\mklvars.bat intel64
   ```

   ```
   nmake arch=intel64 version=offload
   ```
2. Change directory to `bin\intel64`:

   ```
   cd <mkl directory>\benchmarks\mp_linpack\bin\intel64
   ```

   This directory contains files:

   - `xhpl.exe` - the Intel® 64 architecture binary.
   - `HPL.dat` - the HPL input data set.
3. Execute the binary for a small test run:

   ```
   xhpl
   ```
4. Modify the `HPL.dat` file to match the memory on the host processor by increasing the value in line 6 before `Ns`:

   - For 16 GB: `12000 Ns`
   - For 32 GB: `56000 Ns`
   - For 64 GB: `83000 Ns`

   In general, you can compute the memory required to store the matrix (which does not count numerous buffers) as 8 * *N* * *N* / (*P* * *Q*) bytes, where *N* is the problem size, and *P* and *Q* are the process grids in `HPL.dat`. HPL documentation generally recommends choosing a problem size that fills 80% of memory, but you can sometimes use more.
5. Execute the binary again and take note of the new result.

   ```
   xhpl
   ```

For specifics of running hybrid offload binaries, see Running Hybrid Offload Binaries.

### See Also

Notational Conventions
Building the Intel Optimized MP LINPACK Benchmark
Expanding the Benchmark to Two or More Nodes

## Expanding the Benchmark to Two or More Nodes

To run the Intel Optimized MP LINPACK Benchmark on multiple nodes, you need to use MPI and either modify the `HPL.dat` or use Ease-of-use Command-line Parameters as explained in this section.

> **NOTE**
> While these instructions assume the Intel® 64 architecture, they are more widely applicable. The instructions directly apply to Previous Generation Intel® Core™ or higher Intel® processors. For IA-32 architecture processors and for earlier Intel® 64 architecture processors, omit the `version` parameter of the `make` command. For IA-32 architecture processors, also adjust directory names and the value of the `arch` parameter.

To expand runs of the Intel Optimized MP LINPACK Benchmark to more nodes, perform these steps:

1. Load the necessary environment variables for Intel MKL, Intel MPI, and the Intel® compiler and build the binary:

   ```
   <parent directory>\bin\compilervars.bat intel64
   ```

   ```
   <mpi directory>\bin64\mpivars.bat
   ```

   ```
   <mkl directory>\bin\mklvars.bat intel64
   ```

   ```
   nmake arch=intel64 version=offload
   ```
2. Change directory to `bin\intel64`:

   ```
   cd <mkl directory>\benchmarks\mp_linpack\bin\intel64
   ```

   This directory contains files:

   - `xhpl.exe` - the Intel® 64 architecture binary.
   - `HPL.dat` - the HPL input data set.
3. In the `HPL.dat` file, set the problem size `N` to 10000. Because this setting is for a test run, the problem size should be small.
4. In the `HPL.dat` file, set the parameters `Ps` and `Qs` so that `Ps * Qs` equals the number of nodes. For example, for 2 nodes, set `Ps` to 1 and `Qs` to 2. It is easier to achieve optimal result if `Ps = Qs`, so choose them as close to each other as possible so that `Ps ≤ Qs`.

   The resulting `HPL.dat` file for 2 nodes is as follows:

   ```
   HPLinpack benchmark input file
   Innovative Computing Laboratory, University of Tennessee
   HPL.out      output file name (if any)
   6            device out (6=stdout,7=stderr,file)
   1            # of problems sizes (N)
   10000        Ns
   1            # of NBs
   1280         NBs
   1            PMAP process mapping (0=Row-,1=Column-major)
   1            # of process grids (P x Q)
   1            Ps
   2            Qs
   16.0         threshold
   1            # of panel fact
   2            PFACTs (0=left, 1=Crout, 2=Right)
   1            # of recursive stopping criterium
   4            NBMINs (>= 1)
   1            # of panels in recursion
   2            NDIVs
   1            # of recursive panel fact.
   1            RFACTs (0=left, 1=Crout, 2=Right)
   1            # of broadcast
   0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
   ```

```
1            # of lookahead depth
1            DEPTHs (>=0)
0            SWAP (0=bin-exch,1=long,2=mix)
1            swapping threshold
1            L1 in (0=transposed,1=no-transposed) form
1            U  in (0=transposed,1=no-transposed) form
0            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)

Alternatively, launch with -n, -p, and -q parameters and leave the HPL.dat file as is.
```

The resulting `HPL.dat` file for 4 nodes is as follows:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
10000        Ns
1            # of NBs
1280         NBs
1            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
2            Ps
2            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
0            SWAP (0=bin-exch,1=long,2=mix)
1            swapping threshold
1            L1 in (0=transposed,1=no-transposed) form
1            U  in (0=transposed,1=no-transposed) form
0            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)

Alternatively, launch with -n, -p, -q parameters and leave the HPL.dat file as is.
```

5. Run the `xhpl` binary under MPI control on two nodes:

   `mpirun --perhost 1 -n 2 -hosts Node1,Node2`
6. Rerun the HPL test increasing the size of the problem until the matrix size uses about 80% of the available memory. To do this, either modify `Ns` in `HPL.dat` or use the `-m` command-line parameter.

For specifics of running hybrid offload binaries, see Running Hybrid Offload Binaries.

## Running Hybrid Offload Binaries

Hybrid offload binaries of the Intel Optimized MP LINPACK Benchmark inform you how many Intel Xeon Phi coprocessors are detected during the run. Top of the output has a line like this:

`Number of Intel Xeon Phi coprocessors: 1`

**NOTE**
This number counts only one Intel Xeon Phi coprocessor per MPI process.

If Intel Xeon Phi coprocessors are available on your cluster and you expect offloading to occur, but the number printed is zero, it is likely that the correct compiler environment was not loaded.

You can use environment variables specific to hybrid offload binaries to adjust the behavior of your runs. For a list of supported environment variables, see Environment Variables for the Hybrid Offload. Hybrid offload binaries also react to `MKL_MIC_ENABLE` and `OFFLOAD_DEVICES` environment variables for automatic offload to Intel Xeon Phi coprocessors.

The script `runme_intel64.bat` can set the HPL environment variables (such as `HPL_MIC_DEVICE` and `HPL_MIC_SHAREMODE`) for a given numbers of MPI ranks per node and Intel Xeon Phi coprocessors per node so that MPI ranks share the Intel Xeon Phi coprocessors optimally. Set the following variables at the top of these scripts according to your cluster configuration:

`MPI_PROC_NUM` The total number of MPI processes.

`MPI_PER_NODE` The number of MPI processes per each cluster node.

> **TIP**
> To get best performance of HPL, enable non-uniform memory access (NUMA) and set `MPI_PER_NODE` equal to the number of NUMA sockets.

`NUMMIC` The number of Intel Xeon Phi coprocessors per each cluster node.

The scripts launch the hybrid offload HPL binary for a given number of MPI processes and write the results to the output file.

| **Optimization Notice** |
|---|
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

### See Also

Usage Modes of Intel Optimized MP LINPACK Benchmark for Intel® Xeon Phi™ Coprocessors
Running the Benchmark on One Node
Expanding the Benchmark to Two or More Nodes
Automatic Offload Controls

## Improving Performance of Your Cluster

While it is relatively easy to get high performance of the HPL test on a single node, it is more complicated in a cluster. To achieve high performance of the test in a cluster, follow these steps, provided all the needed installations are done on each node:

1. Reboot all nodes.
2. Ensure all nodes are in identical conditions and no zombie processes are left running from prior HPL runs.

To do this, run single-node Stream and HPL LINPACK or the Intel MKL program `nodeperf` on every node. For more details, see also Optimizing the Result on a Cluster. Ensure results are within 2% of each other (problem size must be large enough depending on memory size and CPU speed). Investigate nodes with low performance for hardware/software problems.

3. Check that your cluster interconnects are working. Run a test over the complete cluster using an MPI test for bandwidth and latency.
4. Run an HPL benchmark on pairs of two or four nodes and ensure results are within 4% of each other. The problem size must be large enough depending on the memory size and CPU speed (for example, refer here).
5. Run a small HPL workload over the complete cluster to ensure correctness.
6. Increase the problem size and run the real test load. Rerun at the real size at least three times.
7. In case of problems go back to step 2.

**WARNING**
Each step is important, so skipping a step like 4, you can waste a lot of time.

Before making a heterogeneous inter-node run, always run its homogeneous equivalent first. If you are using Intel Xeon Phi coprocessors in an offload mode for intra-node heterogeneity, first run on the Intel Xeon processors alone.



## More Details of the Intel Optimized MP LINPACK Benchmark

The Intel Optimized MP LINPACK Benchmark does Gaussian elimination with row pivoting to compute an LU decomposition of the matrix. If $P$ is a permutation matrix representing row pivots, then $PA = LU$ where $L$ is a lower unit triangular matrix and $U$ is an upper triangular matrix. The algorithm is blocked to increase cache reuse of the data. The sequential algorithm closest to the Intel Optimized MP LINPACK Benchmark is `DGETRF` from LAPACK or `PDGETRF` from ScaLAPACK, referred to by the generic name `*GETRF`. However, `*GETRF` retains $L$, which is not necessary in the Intel Optimized MP LINPACK Benchmark. A system of equations $Ax = b$ can be solved with `*GETRF` by performing these steps:

1. Compute $PA = LU$
2. Solve $Ly=Pb$ for $y$
3. Solve $Ux = y$ for $x$

$L$ to the left can be discarded if $b$ is replaced with $y$ above while going through the problem. This saves a forward solve at the end (which is not so critical), and it means that as long as you do an LU decomposition on the column-augmented system $[A|b]$, when doing pivots you can skip pivoting to the left while using a right-looking algorithm.

**NOTE**
While it is acceptable in the Intel Optimized MP LINPACK Benchmark to skip pivoting to the left, the LAPACK and ScaLAPACK `*GETRF` algorithms must add pivoting to the left if step 2 might be used later.

`*GETRF` makes several BLAS calls. Assuming *N* is the problem size and *NB* is the column block size used in the blocking above, then as long as *N* is sufficiently greater than *NB*, most of the floating-point operations (FLOPs) are found in `*GEMM`. Some FLOPs may also be present in `*TRSM`. Although other BLAS calls may be necessary, these are the performance critical functions. `*GETRF` does computation in one of three spots: `*GEMM`, `*TRSM`, and a local LU factorization.

`*GEMM` overwrites *C* with
* *op(A)* * *op(B)* + β* *C*.
and β are both scalars, and *A*, *B*, and *C* are matrices. *op(A)* denotes *A* or the transpose of *A* and similarly for *op(B)*. *op(A)* is an *m* x *k* matrix, *op(B)* is a *k* x *n* matrix, and *C* is an *m* x *n* matrix.

Assuming that *N* is the global problem size and that the matrix is distributed on a 2-dimensional (2D) block cyclic mapping on a *P* by *Q* processor grid with the block size *NB*, initial values of *m*, *n*, and *k* for the first few `*GEMM` calls are $m \approx N/P$, $n \approx N/Q$, and $k = NB$.

The number of FLOPs executed in `*GEMM` for each block iteration starts off at approximately 2**N***N***NB*/(*P***Q*). The size of each `*GEMM` may decrease, depending on whether the node in question owns the previous block row or block column.

`*TRSM` solves a triangular system of equations of size *NB* by *NB*, with typically *N/Q* solution vectors to compute. It has roughly *NB*NB*(N/Q)* FLOPs. So, as long as *N* >> *P*NB*, there is more work in `*GEMM` than in `*TRSM`.

Additionally, each iteration does a local LU-factorization that starts at the size *N/P* by *NB*. Although the row pivoting along the entire column has to be done, the computation is the same as factoring an *NBxNB* matrix, which is approximately 2**NB***NB***NB*/3 FLOPs. If *N* >> *P*NB*, there are fewer FLOPs in this computation than in `*TRSM`.

HPL requires row pivoting, which, while it does not involve FLOPs, can still be time consuming. It is not acceptable to replace the random matrix generator of HPL with a matrix that requires less pivoting (for example, a diagonally dominant matrix).

It is also necessary to get the data around the 2D *PxQ* grid since each node needs this data to do the work for row pivoting. Each node in the *PxQ* grid can be represented as the pair (*i*, *j*) where $0 \le i < P$, and $0 \le j < Q$.

The algorithm involves a horizontal broadcast and a vertical broadcast across the grid.

The horizontal broadcast (which contains pivot information and the *A* matrix of the above `*GEMM`) is usually ring broadcast along *Q* nodes. For example, node (2,3) has to get data to every other node in the block row given by (2, *j*). The reason for using a ring broadcast, as opposed to a tree broadcast, is that horizontal motion in the grid can often be overlapped with other operations. The broadcast itself consumes time, but some of this effect can be hidden, depending on the input parameters.

The vertical broadcast (which contains the *B* matrix of the above `*GEMM`) is usually a tree broadcast along *P* nodes because processor columns must be synchronized. In many HPL configurations, $P \le 2Q$ is chosen.

For details of broadcasting in HPL, see www.netlib.org/benchmark/hpl/algorithm.html#bcast.

The other communication is the pivoting itself.

The offload portion of the code is typically some piece of the `*GEMM` or `*TRSM` because they have the highest ratio of FLOPs to data and data must move across the PCIe bus.

The fastest way to offload `DGEMM` is to send some portion of *A* and *B* to the coprocessor and have the coprocessor return *C* (assuming that β was zero) in an Intel Xeon Phi coprocessor native `DGEMM` call.

*NB* must be larger than is required for Intel® Xeon® processors alone. Choosing 400 for *NB* offers no acceleration through offloading. However, choosing a value larger than 960 for *NB* has proven to offer considerable acceleration.

Large values of *NB* require extra memory on the host processor and coprocessor. If this memory is low, the problem size *N* does not satisfy the inequality *N* >> *NB*. In that event, it is better not to use the offload version of the Intel Optimized MP LINPACK Benchmark.

| Optimization Notice |
| --- |
| effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

## Configuring the Hybrid Offload Version of the Intel Optimized MP LINPACK Benchmark

The most significant parameters in `HPL.dat` are *N, NB, P* and *Q*. Specify them as follows, as well as some other parameters:

- *P* and *Q* - the number of rows and columns in the process grid, respectively.

  *P*Q* must be the number of MPI processes that HPL is using.

  For the hybrid offload version of the Intel Optimized MP LINPACK Benchmark, keep *P* and *Q* roughly the same size.

  > **NOTE**
  > This setting is different from settings recommended for most HPL implementations, which usually recommend to choose $P < Q$ and possibly with the 1:4 ratio.

- *NB* - block size of the data distribution.

  The table below shows recommended values of *NB* for 2nd and 3rd generation Intel® Core™ processors and for different numbers of Intel Xeon Phi coprocessors. The values may vary and depend on the PCI Express settings and performance of main memory.

  | No coprocessors | 1 coprocessor | 2 coprocessors | 3 coprocessors |
  | --- | --- | --- | --- |
  | 256 | 960 | 1024 | 1200 |

- *N* - the problem size:
  - For homogeneous runs, choose *N* divisible by *NB*LCM(P,Q)*, where *LCM* is the least common multiple of the two numbers.
  - For heterogeneous runs, refer to Heterogeneous Intel Optimized MP LINPACK Benchmark.

  Note that increasing *N* usually increases performance, but the size of *N* is bounded by memory.
- Other parameters.

  To use Intel MKL BPUSH algorithm for horizontal broadcast, in line 23 of `HPL.dat`, set the `BCASTs` parameter to 6.

Enabling NUMA on your system and running an MPI process for each NUMA socket usually improves the hybrid offload performance. Refer to your BIOS settings for enabling NUMA on your system. You can use `HPL_MIC_DEVICE` and `HPL_MIC_SHAREMODE` environment variables to share the Intel Xeon Phi coprocessors among MPI processes (see Environment Variables for the Hybrid Offload for details). The script`runme_intel64.bat` sets these environment variables for you for a given number of MPI ranks per node.

## Environment Variables for the Hybrid Offload

The table below lists Intel MKL environment variables to control runs of the Intel Optimized MP LINPACK Benchmark in the hybrid offload mode. Each of these environment variables has a default value. Use them with hybrid builds of the benchmark.

**NOTE**
While these environment variables impact performance of the hybrid offload binaries, they do not impact performance of other binaries. Environment variables for automatic offload, listed in Automatic Offload Controls, do not impact performance or behavior of the hybrid offload binaries.

| Environment Variable | Description | Value |
|---|---|---|
| `HPL_LARGEPAGE` | Defines the memory mapping to be used for both the Intel Xeon processor and Intel Xeon Phi coprocessors. | 0 or 1:<br>• 0 - normal memory mapping, default.<br>• 1 - memory mapping with large pages (2 MB per page mapping). It may increase performance. |
| `HPL_LOG` | Controls the level of detail for the HPL output. | An integer ranging from 0 to 2:<br>• 0 - no log is displayed (printed).<br>• 1 - only one root node displays a log, exactly the same as the `ASYOUGO` option provides.<br>• 2 - the most detailed log is displayed. All *P* root nodes in the processor column that owns the current column block display a log. |
| `HPL_HOST_CORE`, `HPL_HOST_NODE` | Specifies cores or Non-Uniform Memory Access (NUMA) nodes to be used.<br>`HPL_HOST_NODE` requires NUMA mode to be enabled. You can check whether it is enabled by the `numactl --hardware` command.<br>The default behavior is auto-detection of the core or NUMA node. | A list of integers ranging from 0 to the largest number of a core or NUMA node in the cluster and separated as explained in example 3. |
| `HPL_SWAPWIDTH` | Specifies width for each swap operation. | 16 or 24. The default is 24. |
| `HPL_MIC_DEVICE` | Specifies Intel Xeon Phi coprocessor(s) to be used. All available Intel Xeon Phi coprocessors are used by default.<br>**NOTE**<br>To avoid oversubscription of resources that might occur if you use multiple MPI processes per node, set this environment variable to specify which coprocessors each MPI process should use. | A comma-separated list of integers, each ranging from 0 to the largest number of an Intel Xeon Phi coprocessor on the node. |

| Environment Variable | Description | Value |
|---|---|---|
| HPL_PNUMMICS | Specifies the number of Intel Xeon Phi coprocessors to be used. The HPL_MIC_DEVICE environment variable takes precedence over HPL_PNUMMICS, and the value of HPL_PNUMMICS is ignored if you set HPL_MIC_DEVICE.<br>The default behavior is auto-detection of the number of coprocessors. | An integer ranging from 0 to the number of Intel Xeon Phi coprocessors in the node. If the value is 0, the core ignores all Intel Xeon Phi coprocessors. |
| HPL_MIC_CORE, HPL_MIC_NODE | Specifies which CPU core will be used for an Intel Xeon Phi coprocessor. Each Intel Xeon Phi coprocessor needs a dedicated CPU core. By setting these variables for an Intel Xeon Phi coprocessor, you reserve:<br>• HPL_MIC_CORE - a specific core.<br>• HPL_MIC_NODE - one of the cores on the specified NUMA node.<br>While the default for HPL_MIC_CORE is some core, the default for HPL_MIC_NODE is a core that the coprocessor shares with the same NUMA node. | An integer ranging from 0 to the largest number of a core or NUMA node for the coprocessor.<br>Can be provided in a comma-separated list, each integer corresponding to one coprocessor. |
| HPL_MIC_NUMCORES | Number of cores to be used for an Intel Xeon Phi coprocessor. All the coprocessor cores are used by default, which produces best performance. | An integer ranging from 1 to the number of cores of the coprocessor. |
| HPL_MIC_SHAREMODE | Specifies whether and how an Intel Xeon Phi coprocessor is shared among two MPI processes.<br>See example 5 for details. | An integer ranging from 0 to 2:<br>• 0 - no sharing, default<br>• 1 - the lower half of the cores will be used for the MPI process.<br>• 2 - the upper half of the cores will be used for the MPI process. |
| HPL_MIC_EXQUEUES | Specifies the queue size on an Intel Xeon Phi coprocessor. Using a larger number is typically better while it increases the memory consumption for the Intel Xeon Phi coprocessor. If out of memory errors are encountered, try a lower number. | An integer ranging from 0 to 512. The default is 128. |

| Environment Variable | Description | Value |
| --- | --- | --- |
| HPL_MIC_WIDTH | Computation width for Intel Xeon Phi DGEMM/DTRSM. If the Intel Xeon Phi coprocessor memory is insufficient, change the settings as follows:<br><br>**1.** Reduce HPL_MIC_WIDTH to 16.<br><br>**NOTE**<br>This might reduce performance of the node.<br><br>**2.** If Intel Xeon Phi coprocessor still reports a memory allocation error:<br>• Reduce the value of the HPL_MIC_EXQUEUES environment variable.<br>• If the node has more than two Intel Xeon Phi coprocessors, use twice larger *P*, the number of rows in the processor grid.<br>**3.** If memory allocation errors are still reported, keep reducing the problem size *N* until the errors are no longer reported. | 16 or 24. The default is 24. |

You can set HPL environment variables using the PMI_RANK and PMI_SIZE environment variables of the Intel MPI library, and you can create a shell script to automate the process.

## Examples of Environment Settings for Hybrid Offload

| # | Settings | Behavior of the Intel Optimized MP Linpack Benchmark |
| --- | --- | --- |
| 1 | Nothing specified | xhpl uses all Intel Xeon processors and all Intel Xeon Phi coprocessors in the cluster. |
| 2 | HPL_PNUMMICS=0 | xhpl ignores Intel Xeon Phi coprocessors and works as a regular HPL. |
| 3 | HPL_MIC_DEVICE=0,2<br><br>HPL_HOST_CORE=1-3,8-10 | Only Intel Xeon Phi coprocessors 0 and 2 and Intel Xeon processor cores 1,2,3,8,9, and 10 are used. |
| 4 | HPL_HOST_NODE=1 | Only cores on NUMA node 1 are used. |
| 5 | HPL_MIC_DEVICE=0,1<br><br>HPL_MIC_SHAREMODE=0,2 | Only Intel Xeon Phi coprocessors 0 and 1 are used:<br>• On the coprocessor 0, all cores are used.<br>• On the coprocessor 1, the upper half of the cores is used.<br>For a 61-core Intel Xeon Phi coprocessor, the upper half includes cores 31-61. |

| # | Settings | Behavior of the Intel Optimized MP Linpack Benchmark |
|---|---|---|
| | | This setting is useful to share an Intel Xeon Phi coprocessor among two MPI processes for an odd number of Intel Xeon Phi coprocessors. |

### See Also

Compile Options Specific to the Intel Optimized MP LINPACK Benchmark

## Optimizing the Result on a Cluster

To benchmark a cluster, follow the sequence of steps below (some of them are optional). Pay special attention to the iterative steps 3 and 4. They make a loop that searches for HPL parameters (specified in `HPL.dat`) that enable you to maximize the performance of your cluster.

1. Install HPL and make sure HPL is functional on all the nodes.
2. (Optional) Run `nodeperf.c` (included in the distribution) to see the performance of `DGEMM` on all the nodes.

   Compile `nodeperf.c` with your MPI and Intel MKL. For example:

   ```
   icl /Za /O3 /w /D_WIN_ /I"<Home directory of MPI libraries>\include" "<Home directory of MPI>\<MPI library>"
   "<mkl directory>\lib\intel64\mkl_core.lib"
   "<parent directory>\lib\intel64\libiomp5md.lib" nodeperf.c
   ```

   where *`<MPI library>`* is `msmpi.lib` in the case of Microsoft* MPI and `mpi.lib` in the case of MPICH.

   Launching `nodeperf` on all the nodes is especially helpful in a very large cluster.`nodeperf` enables quick identification of a potential problem spot without numerous small runs of the Intel Optimized MP LINPACK Benchmark around the cluster in search of a bad node. It goes through all the nodes, one at a time, and reports the performance of `DGEMM` followed by the host identifier. Therefore, the higher the `DGEMM` performance, the faster that node was performing.
3. Edit `HPL.dat` to fit your cluster needs.
   See the HPL documentation for more information. Note, however, that you should use at least 4 nodes.
4. Make an HPL run, using compile options such as `ASYOUGO`, `ASYOUGO2`, or `ENDEARLY` to aid in your search. These options enable you to gain insight into the performance sooner than HPL would normally give this insight.

   When doing so, follow these recommendations:

   - Use the Intel Optimized MP LINPACK Benchmark, which is a patched version of HPL, to save time in the search.

     All the features impacting performance are optional in the Intel Optimized MP LINPACK Benchmark. That is, if you do not use the new options to reduce search time, these features are disabled. The primary purpose of the additions is to assist you in finding solutions.

     While HPL requires a long time to search for many different parameters, in the Intel Optimized MP LINPACK Benchmark, the goal is to get the best possible number.

     Given that the input is not fixed, there is a large parameter space you must search over. An exhaustive search of all possible inputs is improbably large even for a powerful cluster. The Intel Optimized MP LINPACK Benchmark optionally prints information on performance as it proceeds. You can also terminate early.
   - Save time by compiling with `-DENDEARLY-DASYOUGO2` and using a negative threshold (do not use a negative threshold on the final run that you intend to submit as a TOP500 entry). Set the threshold in line 13 of the HPL 2.1 input file `HPL.dat`.
   - If you are going to run a problem to completion, do it with `-DASYOUGO`.
5. Using the quick performance feedback, return to step 3 and iterate until you are sure that the performance is as good as possible.

**See Also**

Options to Reduce Search Time
Notational Conventions

## Options to Reduce Search Time

Running large problems to completion on large numbers of nodes can take many hours. The search space for the Intel Optimized LINPACK Benchmark is also large: you can vary several parameters to improve performance, such as problem size, block size, grid layout, lookahead steps, factorization methods, and so on. You might not want to run a large problem to completion only to discover that it ran 0.01% slower than your previous best problem.

Use the following options to reduce the search time:

- `-DASYOUGO`
- `-DENDEARLY`,
- `-DASYOUGO2`,

  Use `-DASYOUGO2` cautiously because it has a marginal performance impact. To see `DGEMM` internal performance, compile with `-DASYOUGO2` and `-DASYOUGO2_DISPLAY`. These options provide useful `DGEMM` performance information at the cost of around 0.2% performance loss.

If you want to use the original HPL, simply omit these options and recompile from scratch. To do this, try `"nmake arch=<arch> clean_arch_all"`.

### -DASYOUGO

`-DASYOUGO` gives performance data as the run proceeds. The performance always starts off higher and then drops because the LU decomposition slows down as it goes. So the `ASYOUGO` performance estimate is usually an overestimate, but it gets more accurate as the problem proceeds. The greater the lookahead step, the less accurate the first number may be. `ASYOUGO` tries to estimate where execution is in the LU decomposition that Intel Optimized LINPACK Benchmark performs, and this is always an overestimate as compared to `ASYOUGO2`, which measures actually achieved `DGEMM` performance. Note that the `ASYOUGO` output is a subset of the information that `ASYOUGO2` provides. Refer to the description of the `DASYOUGO2` option below for the details of the output.

### -DENDEARLY

`-DENDEARLY` terminates the problem after a few steps, so that you can set up 10 or 20 HPL runs without monitoring them, see how they all do, and then only run the fastest ones to completion. `-DENDEARLY` assumes `-DASYOUGO`. You can define both, but it is not necessary. To avoid the residual check for a problem that terminates early, set the `threshold` parameter in `HPL.dat` to a negative number when testing `ENDEARLY`. It also sometimes gives more information to compile with `-DASYOUGO2` when using `-DENDEARLY`.

Usage notes on `-DENDEARLY` follow:

- `-DENDEARLY` stops the problem after a few iterations of `DGEMM` on the block size (the bigger the blocksize, the further it gets). It prints only five or six updates, whereas `-DASYOUGO` prints about 46 or so output elements before the problem completes.
- Performance for `-DASYOUGO` and `-DENDEARLY` always starts off at one speed, slowly increases, and then slows down toward the end (reflecting the progress of the LU decomposition). `-DENDEARLY` is likely to terminate before it starts to slow down.
- `-DENDEARLY` terminates the problem early with an HPL Error exit. It means that you need to ignore the missing residual results, which are wrong because the problem never completed. However, you can get an idea what the initial performance was, and if it is acceptable, run the problem to completion without `-DENDEARLY`. To avoid the error check, you can set the `threshold` parameter in `HPL.dat` to a negative number.
- Though `-DENDEARLY` terminates early, HPL treats the problem as completed and computes GFLOP rating as though the problem ran to completion. Ignore this erroneously high rating.

- The bigger the problem, the more accurately the last update that -DENDEARLY returns is close to what happens when the problem runs to completion. -DENDEARLY is a poor approximation for small problems. It is for this reason that you should use ENDEARLY in conjunction with ASYOUGO2, because ASYOUGO2 reports actual DGEMM performance, which can be a closer approximation to problems just starting.

## -DASYOUGO2

-DASYOUGO2 gives detailed single-node DGEMM performance information. It captures all DGEMM calls (if you use Fortran BLAS) and records their data. Because of this, the routine has a marginal performance overhead. Unlike -DASYOUGO, which does not impact performance, -DASYOUGO2 interrupts every DGEMM call to monitor its performance. You should be aware of this overhead, although for big problems, it is less than 0.1%.

A sample ASYOUGO2 output appears as follows:

```
Col=001280 Fract=0.050 Mflops=42454.99 (DT=9.5 DF=34.1 DMF=38322.78).
```

> **NOTE**
> The values of Col, Fract, and Mflops are also produced for ASYOUGO and ENDEARLY.

In this example, the problem size is 16000 and a block size is 128. After processing 10 blocks, or 1280 columns (Col), an output was sent to the screen. Here, the fraction of columns completed (Fract) is 1280/16000=0.08. Only up to 111 outputs are printed, at various places through the matrix decomposition:

```
0.005 0.010 0.015 0.020 0.025 0.030 0.035 0.040 0.045 0.050 0.055 0.060 0.065 0.070
0.075 0.080 0.085 0.090 0.095 0.100 0.105 0.110 0.115 0.120 0.125 0.130 0.135 0.140
0.145 0.150 0.155 0.160 0.165 0.170 0.175 0.180 0.185 0.190 0.195 0.200 0.205 0.210
0.215 0.220 0.225 0.230 0.235 0.240 0.245 0.250 0.255 0.260 0.265 0.270 0.275 0.280
0.285 0.290 0.295 0.300 0.305 0.310 0.315 0.320 0.325 0.330 0.335 0.340 0.345 0.350
0.355 0.360 0.365 0.370 0.375 0.380 0.385 0.390 0.395 0.400 0.405 0.410 0.415 0.420
0.425 0.430 0.435 0.440 0.445 0.450 0.455 0.460 0.465 0.470 0.475 0.480 0.485 0.490
0.495 0.515 0.535 0.555 0.575 0.595 0.615 0.635 0.655 0.675 0.695 0.795 0.895.
```

However, this problem size is so small and the block size so big by comparison that as soon as it prints the value for 0.045, it was already through 0.08 fraction of the columns. On a really big problem, the fractional number is more accurate.

-DASYOUGO2 never prints more than the 112 numbers above. So, smaller problems have fewer than 112 updates, and the biggest problems have precisely 112 updates.

Mflops is an estimate based on the number of columns of the LU decomposition being completed. However, with lookahead steps, sometimes that work is not actually completed when the output is made. Nevertheless, this is a good estimate for comparing identical runs.

The three parenthesized numbers are ASYOUGO2 add-ins that impact performance. DT is the total time that processor 0 has spent in DGEMM. DF is the number of billion operations that have been performed in DGEMM by one processor. Therefore, the performance of processor 0 (in GFLOPs) in DGEMM is always DF/DT. Using the number of DGEMM FLOPs as a basis instead of the number of LU FLOPs, you get a lower bound on performance of the run by looking at DMF, which can be compared to Mflops above.

Note that when using the performance monitoring tools described in this section to compare different HPL.dat input data sets, you should be aware that the pattern of performance drop-off that LU experiences is sensitive to sizes of input data. For instance, when you try very small problems, the performance drop-off from the initial values to end values is very rapid. The larger the problem, the less the drop-off, and it is probably safe to use the first few performance values to estimate the difference between a problem size 700000 and 701000, for instance. Another factor that influences the performance drop-off is the relationship of the grid dimensions (*P* and *Q*). For big problems, the performance tends to fall off less from the first few steps when *P* and *Q* are roughly equal. You can make use of a large number of parameters, such as broadcast types, and change them so that the final performance is determined very closely by the first few steps.

Use of these tools can increase the amount of data you can test.

### See Also

Optimizing the Result on a Cluster

# Intel® Math Kernel Library Language Interfaces Support

A

## Language Interfaces Support, by Function Domain

The following table shows language interfaces that Intel® Math Kernel Library (Intel® MKL) provides for each function domain. However, Intel MKL routines can be called from other languages using mixed-language programming. See Mixed-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/<br>Intel MKL 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 |
| Data Fitting | Yes | Yes |
| Extended Eigensolver | Yes | Yes |

| Function Domain | Fortran interface | C/C++ interface |
|---|---|---|
| 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 MKL 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 MKL 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<br>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`‡ |
| Fast Fourier Transforms | `mkl_dfti.f90` | `mkl_dfti.h`‡ |
| Cluster Fast Fourier Transforms | `mkl_cdft.f90` | `mkl_cdft.h`‡‡ |

| Function Domain/ Purpose | Fortran Include Files | C/C++ Include Files |
|---|---|---|
| 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<br>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 MKL 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® Math Kernel Library (Intel® MKL) 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 MKL Fourier transform functions. These collections correspond to the FFTW versions 2.x and 3.x and the Intel MKL versions 7.0 and later.

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

**Important**
For ease of use, FFTW3 interface is also integrated in Intel MKL.

# Directory Structure in Detail

C

Tables in this section show contents of the Intel(R) Math Kernel Library (Intel(R) MKL) architecture-specific directories.

| Optimization Notice |
| --- |
| Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.<br>Notice revision #20110804 |

## Detailed Structure of the IA-32 Architecture Directories

### Static Libraries in the `lib\ia32_win` 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** | | | |
| `mkl_intel_c.lib` | cdecl interface library | | |
| `mkl_intel_s.lib` | CVF default interface library | PGI* and Compaq* Visual Fortran compilers support | |
| `mkl_blas95.lib` | Fortran 95 interface library for BLAS. Supports the Intel® Fortran compiler | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| `mkl_lapack95.lib` | Fortran 95 interface library for LAPACK. Supports the Intel® Fortran compiler | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| **Threading Layer** | | | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | **Name** | **Installed by Default** |
| mkl_intel_thread.lib | OpenMP threading library for the Intel compilers | | |
| mkl_tbb_thread.lib | Intel® Threading Building Blocks (Intel® TBB) threading library for the Intel compilers | Intel TBB threading support | Yes |
| mkl_pgi_thread.lib | OpenMP threading library for the PGI* compiler | PGI* and Compaq* Visual Fortran compilers support | |
| mkl_sequential.lib | Sequential library | | |
| **Computational Layer** | | | |
| mkl_core.lib | Kernel library for IA-32 architecture | | |

## Dynamic Libraries in the `lib\ia32_win` 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** |
| mkl_rt.lib | Single Dynamic Library to be used for linking | | |
| **Interface Layer** | | | |
| mkl_intel_c_dll.lib | cdecl interface library for dynamic linking | | |
| mkl_intel_s_dll.lib | CVF default interface library for dynamic linking | PGI* and Compaq* Visual Fortran compilers support | |
| **Threading Layer** | | | |
| mkl_intel_thread_dll.lib | OpenMP threading library for dynamic linking with the Intel compilers | | |
| mkl_tbb_thread_dll.lib | Intel TBB threading library for the Intel compilers | Intel TBB threading support | Yes |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_pgi_thread_dll.lib | OpenMP threading library for dynamic linking with the PGI* compiler | PGI* and Compaq* Visual Fortran compilers support | |
| mkl_sequential_dll.lib | Sequential library for dynamic linking | | |
| **Computational Layer** | | | |
| mkl_core_dll.lib | Core library for dynamic linking | | |

## Contents of the redist\ia32_win\mkl 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 |
| mkl_rt.dll | Single Dynamic Library | | |
| **Threading Layer** | | | |
| mkl_intel_thread.dll | Dynamic OpenMP threading library for the Intel compilers | | |
| mkl_tbb_thread.dll | Dynamic Intel TBB threading library for the Intel compilers | Intel TBB threading support | Yes |
| mkl_pgi_thread.dll | Dynamic OpenMP threading library for the PGI* compiler | PGI* and Compaq* Visual Fortran compilers support | |
| mkl_sequential.dll | Dynamic sequential library | | |
| **Computational Layer** | | | |
| mkl_core.dll | Core library containing processor-independent code and a dispatcher for dynamic loading of processor-specific code | | |
| mkl_p4.dll | Pentium® 4 processor kernel | | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_p4m.dll | Kernel library for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) enabled processors | | |
| mkl_p4m3.dll | Kernel library for Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) enabled processors | | |
| mkl_avx.dll | Kernel library for Intel® Advanced Vector Extensions (Intel® AVX) enabled processors | | |
| mkl_avx2.dll | Kernel library for Intel® Advanced Vector Extensions 2 (Intel® AVX2) enabled processors | | |
| mkl_avx512.dll | Kernel library for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) enabled processors | | |
| mkl_vml_p4.dll | Vector Mathematics (VM)/Vector Statistics (VS)/Data Fitting (DF) part of Pentium® 4 processor kernel | | |
| mkl_vml_p4m.dll | VM/VS/DF for Intel® SSSE3 enabled processors | | |
| mkl_vml_p4m2.dll | VM/VS/DF for 45nm Hi-k Intel® Core™2 and Intel Xeon® processor families | | |
| mkl_vml_p4m3.dll | VM/VS/DF for Intel® SSE4.2 enabled processors | | |
| mkl_vml_avx.dll | VM/VS/DF optimized for Intel® AVX enabled processors | | |
| mkl_vml_avx2.dll | VM/VS/DF optimized for Intel® AVX2 enabled processors | | |
| mkl_vml_avx512.dll | VM/VS/DF optimized for Intel® AVX-512 enabled processors | | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_vml_ia.dll | VM/VS/DF default kernel for newer Intel® architecture processors | | |
| libmkl_vml_cmpt.dll | VM/VS/DF library for conditional numerical reproducibility | | |
| libimalloc.dll | Dynamic library to support renaming of memory functions | | |
| **Message Catalogs** | | | |
| 1033\mkl_msg.dll | Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English | | |
| 1041\mkl_msg.dll | Catalog of Intel MKL messages in Japanese. Available only if Intel MKL provides Japanese localization. Please see the Release Notes for this information. | | |

# Detailed Structure of the Intel® 64 Architecture Directories

## Static Libraries in the `lib\intel64_win` 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** | | | |
| mkl_intel_lp64.lib | LP64 interface library for the Intel compilers | | |
| mkl_intel_ilp64.lib | ILP64 interface library for the Intel compilers | | |
| mkl_intel_sp2dp.lib | DEPRECATED. SP2DP interface library for the Intel compilers | SP2DP interface | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | **Name** | **Installed by Default** |
| mkl_blas95_lp64.lib | Fortran 95 interface library for BLAS. Supports the Intel® Fortran compiler and LP64 interface | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| mkl_blas95_ilp64.lib | Fortran 95 interface library for BLAS. Supports the Intel® Fortran compiler and ILP64 interface | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| mkl_lapack95_lp64.lib | Fortran 95 interface library for LAPACK. Supports the Intel® Fortran compiler and LP64 interface | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| mkl_lapack95_ilp64.lib | Fortran 95 interface library for LAPACK. Supports the Intel® Fortran compiler and ILP64 interface | Fortran 95 interfaces for BLAS and LAPACK | Yes |
| **Threading Layer** | | | |
| mkl_intel_thread.lib | OpenMP threading library for the Intel compilers | | |
| mkl_tbb_thread.lib | Intel® Threading Building Blocks (Intel® TBB) threading library for the Intel compilers | Intel TBB threading support | Yes |
| mkl_pgi_thread.lib | OpenMP threading library for the PGI* compiler | PGI* Compiler support | |
| mkl_sequential.lib | Sequential library | | |
| **Computational Layer** | | | |
| mkl_core.lib | Kernel library for the Intel® 64 architecture | | |
| **Cluster Libraries** | | | |
| mkl_scalapack_lp64.lib | ScaLAPACK routine library supporting the LP64 interface | Cluster support | |
| mkl_scalapack_ilp64.lib | ScaLAPACK routine library supporting the ILP64 interface | Cluster support | |
| mkl_cdft_core.lib | Cluster version of FFTs | Cluster support | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_blacs_intelmpi_lp64.lib | LP64 version of BLACS routines supporting Intel® MPI Library | Cluster support | |
| mkl_blacs_intelmpi_ilp64.lib | ILP64 version of BLACS routines supporting Intel MPI Library | Cluster support | |
| mkl_blacs_mpich2_lp64.lib | LP64 version of BLACS routines supporting MPICH2 or higher | Cluster support | |
| mkl_blacs_mpich2_ilp64.lib | ILP64 version of BLACS routines supporting MPICH2 or higher | Cluster support | |
| mkl_blacs_msmpi_lp64.lib | LP64 version of BLACS routines supporting Microsoft* MPI | Cluster support | |
| mkl_blacs_msmpi_ilp64.lib | ILP64 version of BLACS routines supporting Microsoft* MPI | Cluster support | |

## Dynamic Libraries in the `lib\intel64_win` 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 |
| mkl_rt.lib | Single Dynamic Library to be used for linking | | |
| **Interface Layer** | | | |
| mkl_intel_lp64_dll.lib | LP64 interface library for dynamic linking with the Intel compilers | | |
| mkl_intel_ilp64_dll.lib | ILP64 interface library for dynamic linking with the Intel compilers | | |
| **Threading Layer** | | | |
| mkl_intel_thread_dll.lib | OpenMP threading library for dynamic linking with the Intel compilers | | |
| mkl_tbb_thread_dll.lib | Intel TBB threading library for the Intel compilers | Intel TBB threading support | Yes |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_pgi_thread_dll.lib | OpenMP threading library for dynamic linking with the PGI* compiler | PGI* Compiler support | |
| mkl_sequential_dll.lib | Sequential library for dynamic linking | | |
| **Computational Layer** | | | |
| mkl_core_dll.lib | Core library for dynamic linking | | |
| **Cluster Libraries** | | | |
| mkl_scalapack_lp64_dll.lib | ScaLAPACK routine library for dynamic linking supporting the LP64 interface | Cluster support | |
| mkl_scalapack_ilp64_dll.lib | ScaLAPACK routine library for dynamic linking supporting the ILP64 interface | Cluster support | |
| mkl_cdft_core_dll.lib | Cluster FFT library for dynamic linking | Cluster support | |
| mkl_blacs_lp64_dll.lib | LP64 version of BLACS interface library for dynamic linking | Cluster support | |
| mkl_blacs_ilp64_dll.lib | ILP64 version of BLACS interface library for dynamic linking | Cluster support | |

## Contents of the redist\intel64_win\mkl 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 |
| mkl_rt.dll | Single Dynamic Library | | |
| **Threading layer** | | | |
| mkl_intel_thread.dll | Dynamic OpenMP threading library for the Intel compilers | | |
| mkl_tbb_thread.dll | Dynamic Intel TBB threading library for the Intel compilers | Intel TBB threading support | Yes |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_pgi_thread.dll | Dynamic OpenMP threading library for the PGI* compiler | PGI* compiler support | |
| mkl_sequential.dll | Dynamic sequential library | | |
| **Computational layer** | | | |
| mkl_core.dll | Core library containing processor-independent code and a dispatcher for dynamic loading of processor-specific code | | |
| mkl_def.dll | Default kernel for the Intel® 64 architecture | | |
| mkl_mc.dll | Kernel library for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) enabled processors | | |
| mkl_mc3.dll | Kernel library for Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) enabled processors | | |
| mkl_avx.dll | Kernel library optimized for Intel® Advanced Vector Extensions (Intel® AVX) enabled processors | | |
| mkl_avx2.dll | Kernel library optimized for Intel® Advanced Vector Extensions 2 (Intel® AVX2) enabled processors | | |
| mkl_avx512.dll | Kernel library optimized for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) enabled processors | | |
| mkl_vml_def.dll | Vector Mathematics (VM)/Vector Statistics (VS)/Data Fitting (DF) part of default kernel | | |
| mkl_vml_mc.dll | VM/VS/DF for Intel® SSSE3 enabled processors | | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_vml_mc2.dll | VM/VS/DF for 45nm Hi-k Intel® Core™2 and Intel Xeon® processor families | | |
| mkl_vml_mc3.dll | VM/VS/DF for Intel® SSE4.2 enabled processors | | |
| mkl_vml_avx.dll | VM/VS/DF optimized for Intel® AVX enabled processors | | |
| mkl_vml_avx2.dll | VM/VS/DF optimized for Intel® AVX2 enabled processors | | |
| mkl_vml_avx512.dll | VM/VS/DF optimized for Intel® AVX-512 enabled processors | | |
| libmkl_vml_cmpt.dll | VM/VS/DF library for conditional numerical reproducibility | | |
| libimalloc.dll | Dynamic library to support renaming of memory functions | | |
| **Cluster Libraries** | | | |
| mkl_scalapack_lp64.dll | ScaLAPACK routine library supporting the LP64 interface | Cluster support | |
| mkl_scalapack_ilp64.dll | ScaLAPACK routine library supporting the ILP64 interface | Cluster support | |
| mkl_cdft_core.dll | Cluster FFT dynamic library | Cluster support | |
| mkl_blacs_lp64.dll | LP64 version of BLACS routines | Cluster support | |
| mkl_blacs_ilp64.dll | ILP64 version of BLACS routines | Cluster support | |
| mkl_blacs_intelmpi_lp64.dll | LP64 version of BLACS routines for Intel® MPI Library | Cluster support | |
| mkl_blacs_intelmpi_ilp64.dll | ILP64 version of BLACS routines for Intel MPI Library | Cluster support | |
| mkl_blacs_mpich2_lp64.dll | LP64 version of BLACS routines for MPICH2 or higher | Cluster support | |

| File | Contents | Optional Component | |
|---|---|---|---|
| | | Name | Installed by Default |
| mkl_blacs_mpich2_ilp64.dll | ILP64 version of BLACS routines for MPICH2 or higher | Cluster support | |
| mkl_blacs_msmpi_lp64.dll | LP64 version of BLACS routines for Microsoft* MPI | Cluster support | |
| mkl_blacs_msmpi_ilp64.dll | ILP64 version of BLACS routines for Microsoft* MPI | Cluster support | |
| **Message Catalogs** | | | |
| 1033\mkl_msg.dll | Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English | | |
| 1041\mkl_msg.dll | Catalog of Intel MKL messages in Japanese. Available only if Intel MKL provides Japanese localization. Please see the Release Notes for this information. | | |

# Detailed Directory Structure of the `lib/intel64_win_mic` Directory

| File | Contents |
|---|---|
| **Static Libraries** | |
| libmkl_intel_lp64.a | LP64 interface library for the Intel compilers |
| libmkl_intel_ilp64.a | ILP64 interface library for the Intel compilers |
| libmkl_blas95_lp64.a | Fortran 95 interface library for BLAS and Intel® Fortran compiler. Supports the LP64 interface. |
| libmkl_blas95_ilp64.a | Fortran 95 interface library for BLAS and Intel Fortran compiler. Supports the ILP64 interface. |
| libmkl_lapack95_lp64.a | Fortran 95 interface library for LAPACK and Intel Fortran compiler. Supports the LP64 interface. |
| libmkl_lapack95_ilp64.a | Fortran 95 interface library for LAPACK and Intel Fortran compiler. Supports the ILP64 interface. |
| libmkl_intel_thread.a | OpenMP threading library for the Intel compilers |
| libmkl_tbb_thread.a | Intel TBB threading library for the Intel compilers |
| libmkl_sequential.a | Sequential library |
| libmkl_core.a | Core computation library |

| File | Contents |
|---|---|
| `libmkl_scalapack_lp64.a` | Static library with LP64 versions of ScaLAPACK routines |
| `libmkl_scalapack_ilp64.a` | Static library with ILP64 versions of ScaLAPACK routines |
| `libmkl_cdft_core.a` | Static library with cluster FFT functions |
| `libmkl_blacs_intelmpi_lp64.a` | Static library with LP64 versions of BLACS routines for Intel MPI |
| `libmkl_blacs_intelmpi_ilp64.a` | Static library with ILP64 versions of BLACS routines for Intel MPI |
| **Dynamic Libraries** | |
| `libmkl_rt.so` | Single Dynamic Library |
| `libmkl_ao_worker.so` | The Intel® MIC Architecture library to implement the Automatic Offload mode |
| `libmkl_intel_lp64.so` | LP64 interface library for the Intel compilers |
| `libmkl_intel_ilp64.so` | ILP64 interface library for the Intel compilers |
| `libmkl_intel_thread.so` | OpenMP threading library for the Intel compilers |
| `libmkl_tbb_thread.so` | Intel TBB threading library for the Intel compilers |
| `libmkl_sequential.so` | Sequential library |
| `libmkl_core.so` | Core computation library |
| `libmkl_scalapack_lp64.so` | Dynamic library with LP64 versions of ScaLAPACK routines |
| `libmkl_scalapack_ilp64.so` | Dynamic library with ILP64 versions of ScaLAPACK routines |
| `libmkl_cdft_core.so` | Dynamic library with cluster FFT functions |
| `libmkl_blacs_intelmpi_lp64.so` | Dynamic library with LP64 versions of BLACS routines for Intel MPI |
| `libmkl_blacs_intelmpi_ilp64.so` | Dynamic library with ILP64 versions of BLACS routines for Intel MPI |
| `locale/en_US/mkl_msg.cat` | Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English |
| `locale/ja_JP/mkl_msg.cat` | Catalog of Intel MKL messages in Japanese. Available only if Intel MKL provides Japanese localization. Please see the Release Notes for this information. |

# Index

## K



## L



## M



## N



## O



## P



## R



## S