



Intel® Math Kernel Library for Windows*

Developer Guide

Intel® MKL 2017 - Windows*

Revision: 054

Legal Information

Contents

Legal Information.....	9
Getting Help and Support.....	11
Introducing the Intel® Math Kernel Library.....	13
What's New.....	15
Notational Conventions.....	17
Related Information.....	19
 Chapter 1: Getting Started	
Checking Your Installation.....	21
Setting Environment Variables	21
Compiler Support.....	22
Using Code Examples.....	23
What You Need to Know Before You Begin Using the Intel® Math Kernel Library.....	23
 Chapter 2: Structure of the Intel® Math Kernel Library	
Architecture Support.....	27
High-level Directory Structure.....	27
Layered Model Concept.....	29
 Chapter 3: Linking Your Application with the Intel® Math Kernel Library	
Linking Quick Start.....	31
Using the /Qmkl Compiler Option.....	31
Automatically Linking a Project in the Visual Studio* Integrated Development Environment with Intel® MKL.....	32
Automatically Linking Your Microsoft Visual C/C++* Project with Intel® MKL.....	32
Automatically Linking Your Intel® Visual Fortran Project with Intel® MKL.....	32
Using the Single Dynamic Library.....	33
Selecting Libraries to Link with.....	33
Using the Link-line Advisor.....	34
Using the Command-line Link Tool.....	34
Linking Examples.....	34
Linking on IA-32 Architecture Systems.....	34
Linking on Intel(R) 64 Architecture Systems.....	35
Linking in Detail.....	36
Dynamically Selecting the Interface and Threading Layer.....	36
Linking with Interface Libraries.....	38
Using the cdecl and stdcall Interfaces.....	38
Using the ILP64 Interface vs. LP64 Interface.....	39
Linking with Fortran 95 Interface Libraries.....	41
Linking with Threading Libraries.....	41
Linking with Computational Libraries.....	42
Linking with Compiler Run-time Libraries.....	43
Linking with System Libraries.....	44

Building Custom Dynamic-link Libraries.....	44
Using the Custom Dynamic-link Library Builder in the Command-line Mode.....	44
Composing a List of Functions	46
Specifying Function Names.....	46
Building a Custom Dynamic-link Library in the Visual Studio* Development System.....	47
Distributing Your Custom Dynamic-link Library.....	48

Chapter 4: Managing Performance and Memory

Improving Performance with Threading.....	49
OpenMP* Threaded Functions and Problems.....	49
Functions Threaded with Intel® Threading Building Blocks.....	51
Avoiding Conflicts in the Execution Environment.....	52
Techniques to Set the Number of Threads.....	52
Setting the Number of Threads Using an OpenMP* Environment Variable.....	53
Changing the Number of OpenMP* Threads at Run Time.....	53
Using Additional Threading Control.....	56
Intel MKL-specific Environment Variables for OpenMP Threading Control.....	56
MKL_DYNAMIC.....	57
MKL_DOMAIN_NUM_THREADS.....	57
MKL_NUM_STRIPES.....	59
Setting the Environment Variables for Threading Control.....	60
Calling Intel MKL Functions from Multi-threaded Applications.....	60
Using Intel® Hyper-Threading Technology.....	62
Managing Multi-core Performance.....	62
Improving Performance for Small Size Problems	63
Using MKL_DIRECT_CALL in C Applications.....	63
Using MKL_DIRECT_CALL in Fortran Applications.....	64
Limitations of the Direct Call	64
Other Tips and Techniques to Improve Performance.....	65
Coding Techniques.....	65
Improving Intel(R) MKL Performance on Specific Processors.....	66
Operating on Denormals.....	66
Fast Fourier Transform Optimized Radices.....	66
Using Memory Functions	67
Avoiding Memory Leaks in Intel MKL.....	67
Redefining Memory Functions.....	67

Chapter 5: Language-specific Usage Options

Using Language-Specific Interfaces with Intel® Math Kernel Library.....	69
Interface Libraries and Modules.....	69
Fortran 95 Interfaces to LAPACK and BLAS.....	70
Compiler-dependent Functions and Fortran 90 Modules.....	71
Using the stdcall Calling Convention in C/C++.....	71
Compiling an Application that Calls the Intel® Math Kernel Library and Uses the CVF Calling Conventions.....	72
Mixed-language Programming with the Intel Math Kernel Library.....	73

Calling LAPACK, BLAS, and CBLAS Routines from C/C++ Language Environments.....	73
Using Complex Types in C/C++.....	74
Calling BLAS Functions that Return the Complex Values in C/C++ Code..	75
Chapter 6: Obtaining Numerically Reproducible Results	
Getting Started with Conditional Numerical Reproducibility	80
Specifying Code Branches.....	81
Reproducibility Conditions.....	82
Setting the Environment Variable for Conditional Numerical Reproducibility.....	83
Code Examples.....	83
Chapter 7: Coding Tips	
Example of Data Alignment.....	87
Using Predefined Preprocessor Symbols for Intel® MKL Version-Dependent Compilation.....	88
Chapter 8: Managing Output	
Using Intel MKL Verbose Mode.....	91
Version Information Line.....	91
Call Description Line.....	92
Chapter 9: Working with the Intel® Math Kernel Library Cluster Software	
Message-Passing Interface Support.....	95
Linking with Intel MKL Cluster Software.....	96
Determining the Number of OpenMP* Threads.....	97
Using DLLs.....	98
Setting Environment Variables on a Cluster.....	98
Interaction with the Message-passing Interface.....	99
Using a Custom Message-Passing Interface.....	99
Examples of Linking for Clusters.....	100
Examples for Linking a C Application.....	101
Examples for Linking a Fortran Application.....	101
Chapter 10: Using Intel® Math Kernel Library on Intel® Xeon Phi™ Coprocessors	
Automatic Offload.....	104
Automatic Offload Controls.....	104
Setting Environment Variables for Automatic Offload.....	109
Compiler Assisted Offload.....	110
Examples of Compiler Assisted Offload.....	111
Linking for Compiler Assisted Offload.....	112
Using Automatic Offload and Compiler Assisted Offload in One Application.....	113
Running Intel MKL on an Intel Xeon Phi Coprocessor in Native Mode.....	113
Using ScaLAPACK and Cluster FFT on Intel Xeon Phi Coprocessors.....	113
Examples of Linking with ScaLAPACK and Cluster FFT for Intel(R) Many Integrated Core Architecture.....	114
Threading Behavior of Intel MKL on Intel MIC Architecture.....	116
Improving Performance on Intel Xeon Phi Coprocessors	116

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

Managing Behavior of Function Domains.....	119
Setting the Default Mode of Vector Math with an Environment Variable..	119
Managing Performance of the Cluster Fourier Transform Functions.....	120
Instruction Set Specific Dispatching on Intel® Architectures.....	121

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

Configuring Your Integrated Development Environment to Link with Intel(R) Math Kernel Library	123
Configuring the Microsoft Visual C/C++* Development System to Link with Intel® MKL.....	123
Configuring Intel® Visual Fortran to Link with Intel MKL.....	124
Getting Assistance for Programming in the Microsoft Visual Studio* IDE	124
Using Context-Sensitive Help.....	124
Using the IntelliSense* Capability.....	124

Chapter 13: Intel® Math Kernel Library Benchmarks

Intel® Optimized LINPACK Benchmark for Windows*.....	127
Contents of the Intel® Optimized LINPACK Benchmark.....	127
Running the Software.....	128
Known Limitations of the Intel® Optimized LINPACK Benchmark.....	129
Intel® Optimized MP LINPACK Benchmark for Clusters.....	129
Overview of the Intel Optimized MP LINPACK Benchmark.....	129
Usage Modes of Intel Optimized MP LINPACK Benchmark for Intel® Xeon Phi™ Coprocessors and Processors.....	130
Contents of the Intel Optimized MP LINPACK Benchmark.....	131
Building the Intel Optimized MP LINPACK Benchmark for a Customized MPI Implementation.....	131
Building the Netlib HPL from Source Code.....	132
Configuring Parameters.....	132
Ease-of-use Command-line Parameters.....	133
Running the Intel Optimized MP LINPACK Benchmark.....	133
Offloading to Intel Xeon Phi Coprocessors.....	134
Heterogeneous Support in the Intel Optimized MP LINPACK Benchmark.....	135
Environment Variables.....	136
Improving Performance of Your Cluster.....	139

Appendix A: Intel® Math Kernel Library Language Interfaces Support

Language Interfaces Support, by Function Domain.....	141
Include Files.....	142

Appendix B: Support for Third-Party Interfaces

FFTW Interface Support.....	145
-----------------------------	-----

Appendix C: Directory Structure in Detail

Detailed Structure of the IA-32 Architecture Directories.....	147
Static Libraries in the lib\ia32_win Directory.....	147
Dynamic Libraries in the lib\ia32_win Directory.....	148

Contents of the redist\ia32_win\mkl Directory.....	149
Detailed Structure of the Intel® 64 Architecture Directories.....	151
Static Libraries in the lib\intel64_win Directory.....	151
Dynamic Libraries in the lib\intel64_win Directory.....	153
Contents of the redist\intel64_win\mkl Directory.....	154
Detailed Directory Structure of the lib/intel64_win_mic Directory.....	157

Legal Information

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors which may cause deviations from published specifications. Current characterized errata are available on request.

Cilk, Intel, the Intel logo, Intel Atom, Intel Core, Intel Inside, Intel NetBurst, Intel SpeedStep, Intel vPro, Intel Xeon Phi, Intel XScale, Itanium, MMX, Pentium, Thunderbolt, Ultrabook, VTune and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

© 2017, Intel Corporation.

Optimization Notice
<p>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.</p> <p>Notice revision #20110804</p>

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.
- Deep Neural Network (DNN) primitive functions with C language interface.

For details see the *Intel® MKL Developer Reference*.

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 Architecture (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.

The library requires subroutine and function parameters to be valid before being passed. While some Intel MKL routines do limited checking of parameter errors, your application should check for NULL pointers, for example.

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.

Notice revision #20110804

What's New

This Developer Guide documents Intel® Math Kernel Library (Intel® MKL) 2017 Update 2.

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

- Coding techniques to improve performance of an application that calls Intel MKL are explained for Intel® Xeon Phi™ processor x200 product family, codenamed Knights Landing. For more details, see [Coding Techniques](#).
- The `MKL_NUM_STRIPEs` environment variable has been added to control the Intel MKL threading algorithm for `?gemm` functions. For details, see [MKL_NUM_STRIPEs](#).
- All Vector Mathematics functions are now threaded with Intel® Threaded Building Blocks (Intel® TBB). For more details, see [Functions Threaded with Intel Threading Building Blocks](#).
- The instruction set architecture that Intel MKL dispatches by default has changed from Intel® Advanced Vector Extensions 2 to Intel® Advanced Vector Extensions 512. For more details, see [Instruction Set Specific Dispatching on Intel® Architectures](#).

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* 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:
<mkl directory>=<parent directory>\mkl.
Replace this placeholder with the specific pathname in the configuring, linking, and building instructions.

The following font conventions are used in this document.

Italic Italic is used for emphasis and also indicates document names in body text, for example:
see *Intel MKL Developer Reference*.

Monospace lowercase mixed with uppercase Indicates:

- Commands and command-line options, for example,

```
ifort myprog.f mkl_blas95.lib mkl_c.lib libiomp5md.lib
```

- Filenames, directory names, and pathnames, for example,

```
C:\Program Files\Java\jdk1.5.0_09
```

- C/C++ code fragments, for example,

```
a = new double [SIZE*SIZE];
```

UPPERCASE
MONOSPACE Indicates system variables, for example, \$MKLPATH.

Monospace italic Indicates a parameter in discussions, for example, *lda*.
When enclosed in angle brackets, indicates a placeholder for an identifier, an expression, a string, a symbol, or a value, for example, *<mkl directory>*.
Substitute one of these items for the placeholder.

[items] Square brackets indicate that the items enclosed in brackets are optional.

{ item | item } Braces indicate that only one of the items listed between braces should be selected.
A vertical bar (|) separates the items.

Related Information

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

- The *Intel® Math Kernel Library Developer Reference*, 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

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

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* 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 Architecture (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 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 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 `vm1c` 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	<p>Identify the architecture of your target machine:</p> <ul style="list-style-type: none"> • IA-32 or compatible • Intel® 64 or compatible <p>Reason: Because Intel MKL libraries are located in directories corresponding to your particular architecture (see Architecture 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).</p>
Mathematical problem	<p>Identify all Intel MKL function domains that you require:</p> <ul style="list-style-type: none"> • BLAS • Sparse BLAS • LAPACK • PBLAS • ScaLAPACK • Sparse Solver routines • Parallel Direct Sparse Solvers for Clusters

- Vector Mathematics functions (VM)
- Vector Statistics functions (VS)
- Fourier Transform functions (FFT)
- Cluster FFT
- Trigonometric Transform routines
- Poisson, Laplace, and Helmholtz Solver routines
- Optimization (Trust-Region) Solver routines
- Data Fitting Functions
- Extended Eigensolver Functions

Reason: The function domain you intend to use narrows the search in the *Intel MKL Developer Reference* 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](#)).

Reason: Intel MKL provides language-specific include files for each function domain to simplify program development (see [Language Interfaces Support_ by Function Domain](#)).

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

Range of integer data

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

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

Threading model

Identify whether and how your application is threaded:

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

Reason: The compiler you use to thread your application determines which threading library you should link with your application. For applications threaded with a third-party compiler you may need to use Intel 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.

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:

- Static
- Dynamic

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.

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.

Notice revision #20110804

Architecture Support

Intel® Math Kernel Library (Intel® MKL) for Windows* 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	<code><mkl_directory>\lib\ia32_win</code> <code><parent_directory>\redist\ia32_win\mkl</code> (DLLs)
Intel® 64 or compatible	<code><mkl_directory>\lib\intel64_win</code> <code><parent_directory>\redist\intel64_win\mkl</code> (DLLs)
Intel® Many Integrated Core Architecture (Intel® MIC Architecture)	<code><mkl_directory>\lib\intel64_win_mic</code>

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
<code><mkl_directory></code>	Installation directory of the Intel® Math Kernel Library (Intel® MKL)
Subdirectories of <code><mkl_directory></code>	
<code>bin</code>	Batch files to set environmental variables in the user shell

Directory	Contents
bin\ia32	Batch files for the IA-32 architecture
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\mk1	DLLs for applications running on processors with the IA-32 architecture
redist\intel64_win\mk1	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	<p>This layer matches compiled code of your application with the threading and/or computational parts of the library. This layer provides:</p> <ul style="list-style-type: none"> • cdecl and CVF default interfaces. • LP64 and ILP64 interfaces. • Compatibility with compilers that return function values differently.
Threading Layer	<p>This layer:</p> <ul style="list-style-type: none"> • Provides a way to link threaded Intel MKL with supported compilers. • Enables you to link with a threaded or sequential mode of the library. <p>This layer is compiled for different environments (threaded or sequential) and compilers (from Intel and PGI*).</p>
Computational Layer	<p>This layer accommodates multiple architectures through identification of architecture features and chooses the appropriate binary code at run time.</p>

See Also[Using the ILP64 Interface vs. LP64 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	<code>mkl_intel_c.lib</code>	<code>mkl_intel_thread.lib</code>	<code>mkl_core.lib</code>	<code>libiomp5md.lib</code>
IA-32 architecture, dynamic linking	<code>mkl_intel_c_dll.lib</code>	<code>mkl_intel_thread_dll.lib</code>	<code>mkl_core_dll.lib</code>	<code>libiomp5md.lib</code>
Intel® 64 architecture, static linking	<code>mkl_intel_lp64.lib</code>	<code>mkl_intel_thread.lib</code>	<code>mkl_core.lib</code>	<code>libiomp5md.lib</code>
Intel® 64 architecture, dynamic linking	<code>mkl_intel_lp64_dll.lib</code>	<code>mkl_intel_thread_dll.lib</code>	<code>mkl_core_dll.lib</code>	<code>libiomp5md.lib</code>
Intel® Many Integrated Core Architecture (Intel® MIC Architecture), static linking	<code>libmkl_intel_lp64.a</code>	<code>libmkl_intel_thread.a</code>	<code>libmkl_core.a</code>	<code>libiomp5.so</code>
Intel MIC Architecture, dynamic linking	<code>libmkl_intel_lp64.so</code>	<code>libmkl_intel_thread.so</code>	<code>libmkl_core.so</code>	<code>libiomp5.so</code>

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	<code>mkl_rt.lib</code>	<code>libiomp5md.lib</code> [†]

[†]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 <code>MKL_INTERFACE_LAYER</code>	Value of the Parameter of <code>mkl_set_interface_layer</code>
Intel LP64, default	LP64	<code>MKL_INTERFACE_LP64</code>
Intel ILP64	ILP64	<code>MKL_INTERFACE_ILP64</code>

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

See the *Intel MKL Developer Reference* 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 <code>MKL_THREADING_LAYER</code>	Value of the Parameter of <code>mkl_set_threading_layer</code>
Intel threading, default	INTEL	<code>MKL_THREADING_INTEL</code>
Sequential mode of Intel MKL	SEQUENTIAL	<code>MKL_THREADING_SEQUENTIAL</code>
PGI threading [†]	PGI	<code>MKL_THREADING_PGI</code>
Intel TBB threading	TBB	<code>MKL_THREADING_TBB</code>

[†] 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 Developer Reference* 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 Developer Reference* 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
<code>mkl_intel_c.lib</code>	<code>mkl_intel_c_dll.lib</code>	cdecl	At the end
		The defaults of Intel® C++ and Intel® Fortran compilers	
<code>mkl_intel_s.lib</code>	<code>mkl_intel_s_dll.lib</code>	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
<code>mkl_intel_c[_dll].lib</code>	Use the following declaration: <pre><type> name(<prototype variable1>, <prototype variable2>, ..);</pre>
<code>mkl_intel_s[_dll].lib</code>	Call a routine with the following statement: <pre>extern __stdcall name(<prototype variable1>, <prototype variable2>, ..);</pre>

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 or /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 `/QI8` 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:	INTEGER without specifying KIND	MKL_INT
<ul style="list-style-type: none"> 64-bit for ILP64 32-bit otherwise 		
Universal integers for ILP64/ LP64:	INTEGER*8 or INTEGER(KIND=8)	MKL_INT64
<ul style="list-style-type: none"> 64-bit integers 		
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 `vs1SkipAheadStream` service function are 64-bit.
- **Summary Statistics:** The *estimate* parameter of the `vs1sSSCompute/vs1dSSCompute` function is 64-bit.

Refer to the *Intel MKL Developer Reference* 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

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: <code>mkl_intel_thread.lib</code> Dynamic linking: <code>mkl_intel_thread_dll.lib</code>	<code>libiomp5md.lib</code>

Application		Intel MKL		RTL Required
Uses OpenMP	Compiled with	Execution Mode	Threading Layer	
no	any compiler	parallel	Static linking: mkl_tbb_ thread.lib Dynamic linking: mkl_tbb_ thread_dll.lib	tbb.lib
no	any compiler	sequential	Static linking: mkl_ sequential.lib Dynamic linking: mkl_ sequential_dll.lib	none
yes	Intel compiler	parallel	Static linking: mkl_intel_ thread.lib Dynamic linking: mkl_intel_ thread_dll.lib	libiomp5md.lib
yes	PGI* compiler	parallel	Static linking: mkl_pgi_ thread.lib Dynamic linking: 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 mkl_core.lib	mkl_scalapack_core_dll.lib mkl_core_dll.lib
Cluster Fourier Transform Functions [†]	mkl_cdft_core.lib mkl_core.lib	mkl_cdft_core_dll.lib 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 mkl_core.lib	mkl_scalapack_lp64_dll.lib mkl_core_dll.lib
ScaLAPACK, ILP64 interface [‡]	mkl_scalapack_ilp64.lib mkl_core.lib	mkl_scalapack_ilp64_dll.lib mkl_core_dll.lib
Cluster Fourier Transform Functions [‡]	mkl_cdft_core.lib mkl_core.lib	mkl_cdft_core_dll.lib 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
<code>libia32</code>	The builder uses static Intel MKL interface, threading, and core libraries to build a custom DLL for the IA-32 architecture.
<code>libintel64</code>	The builder uses static Intel MKL interface, threading, and core libraries to build a custom DLL for the Intel® 64 architecture.
<code>dllia32</code>	The builder uses the single dynamic library <code>libmkl_rt.dll</code> to build a custom DLL for the IA-32 architecture.
<code>dllintel64</code>	The builder uses the single dynamic library <code>libmkl_rt.dll</code> to build a custom DLL for the Intel® 64 architecture.
<code>help</code>	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
<code>interface</code>	Defines which programming interface to use. Possible values: <ul style="list-style-type: none"> For the IA-32 architecture, <code>{cdecl stdcall}</code>. The default value is <code>cdecl</code>. For the Intel 64 architecture, <code>{lp64 ilp64}</code>. The default value is <code>lp64</code>.
<code>threading = {parallel sequential}</code>	Defines whether to use the Intel MKL in the threaded or sequential mode. The default value is <code>parallel</code> .

Parameter [Values]	Description
<code>export = <file name></code>	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 <code>user_example_list</code> (no extension).
<code>name = <dll name></code>	Specifies the name of the dll and interface library to be created. By default, the names of the created libraries are <code>mkl_custom.dll</code> and <code>mkl_custom.lib</code> .
<code>xerbla = <error handler></code>	Specifies the name of the object file <code><user_xerbla>.obj</code> 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 <code>xerbla</code> . If you omit this parameter, the native Intel MKL <code>xerbla</code> is used. See the description of the <code>xerbla</code> function in the Intel MKL Developer Reference 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 (<code>cdecl</code> or <code>stdcall</code>).
<code>MKLROOT = <mkl directory></code>	Specifies the location of Intel MKL libraries used to build the custom DLL. By default, the builder uses the Intel MKL installation directory.
<code>buf_lib</code>	<p>Manages resolution of the <code>__security_cookie</code> external references in the custom DLL on systems based on the Intel® 64 architecture.</p> <p>By default, the makefile uses the <code>bufferoverflowu.lib</code> library of Microsoft SDK builds 1289 or higher. This library resolves the <code>__security_cookie</code> external references.</p> <p>To avoid using this library, set the empty value of this parameter. Therefore, if you are using an older SDK, set <code>buf_lib=</code> .</p>
CAUTION Use the <code>buf_lib</code> parameter only with the empty value. Incorrect value of the parameter causes builder errors.	
<code>crt = <c run-time library></code>	Specifies the name of the Microsoft C run-time library to be used to build the custom DLL. By default, the builder uses <code>msvcrt.lib</code> .
<code>manifest = {yes no embed}</code>	<p>Manages the creation of a Microsoft manifest for the custom DLL:</p> <ul style="list-style-type: none"> • If <code>manifest=yes</code>, the manifest file with the name defined by the name parameter above and the <code>manifest</code> extension will be created. • If <code>manifest=no</code>, the manifest file will not be created. • If <code>manifest=embed</code>, the manifest will be embedded into the DLL. <p>By default, the builder does not use the <code>manifest</code> parameter.</p>

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

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**.

- a. 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.

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 `zlacon`) 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: ?dttrs, ?gbtrs, ?gttrs, ?pptrs, ?pbtrs, ?pttrs, ?sytrs, ?sptrs, ?hptrs, ?tpttrs, ?tbtrs
- Orthogonal factorization, computational routines:
 - ?geqrf, ?ormqr, ?unmqr, ?ormql, ?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/zggev, 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, α 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).
- All Vector Mathematics functions (except service functions).
- 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:

α geqrf, α gelqf, α getrf, α potrf, α unmqr*, α ormqr*, α unmrq*, α ormrq*, α unmlq*, α ormlq*, α unmql*, α ormql*, α sytrd, α hetrd, α syev, α heev, and α latrd.

A number of other LAPACK routines, which are based on threaded LAPACK or BLAS routines, make effective use of Intel TBB threading:

α getrs, α gesv, α potrs, α bdsqr, and α 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*).	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: <code>mkl_sequential.lib</code> or <code>mkl_sequential.dll</code> (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: <code>mkl_sequential.lib</code> or <code>mkl_sequential_dll.lib</code> (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 Developer Reference* 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 Developer Reference* 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 or Windows* 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);`

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

```

}
free (a);
free (b);
free (c);
return 0;
}

```

```

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

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

```

Using Additional Threading Control

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 Developer Reference* 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	<code>mkl_set_num_threads</code> <code>mkl_set_num_threads_local</code>	Suggests the number of OpenMP threads to use.	OMP_NUM_THREADS
MKL_DOMAIN_NUM_THREADS	<code>mkl_domain_set_num_threads</code>	Suggests the number of OpenMP threads for a particular function domain.	
MKL_DYNAMIC	<code>mkl_set_dynamic</code>	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:

```
// ***** C language *****
#include <mkl.h>
```



```
...
mkl_set_num_threads ( 1 );
```

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

See the *Intel MKL Developer Reference* 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, 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 `mk1.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 );
```

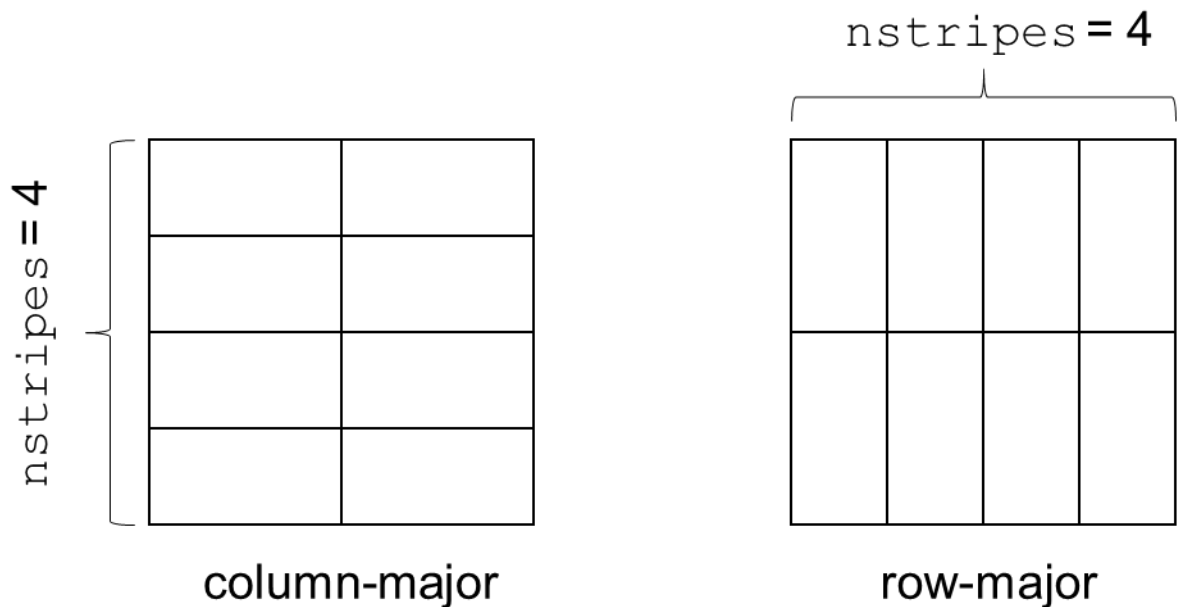
`MKL_NUM_STRIPEs`

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

The following table explains how the value *nstripes* of `MKL_NUM_STRIPEs` defines the partitioning algorithm used by Intel MKL for `?gemm` output matrix; *max_threads_for_mkl* denotes the maximum number of OpenMP threads for Intel MKL:

Value of <code>MKL_NUM_STRIPEs</code>	Partitioning Algorithm
$1 < nstripes < \frac{(max_threads_for_mkl)}{2}$	2D partitioning with the number of partitions equal to <i>nstripes</i> : <ul style="list-style-type: none"> Horizontal, for column-major ordering. Vertical, for row-major ordering.
$nstripes = 1$	1D partitioning algorithm along the opposite direction of the leading dimension.
$nstripes \geq \frac{(max_threads_for_mkl)}{2}$	1D partitioning algorithm along the leading dimension.
$nstripes < 0$	The default Intel MKL threading algorithm.

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



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

Setting the Environment Variables for Threading Control

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

```
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
set MKL_NUM_STRIPES=4
```

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
export MKL_NUM_STRIPES=4
```

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 `mk1_rt.lib` and select the sequential library using an environment variable or a function call:
 - Set `MKL_THREADING_LAYER=sequential`
 - Call `mk1_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 `mk1_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 Developer Reference*, 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.

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 that threads do not migrate from core to core. To do this, bind threads to the CPU cores by setting 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 to appropriate cores, preventing migration of the threads. Then the Intel MKL LAPACK routine is called:

```
// Set affinity mask
#include <windows.h>
#include <omp.h>
int main(void) {
#pragma omp parallel default(shared)
{
int tid = omp_get_thread_num();
// 2 packages x 2 cores/pkg x 1 threads/core (4 total cores)
DWORD_PTR mask = (1 << (tid == 0 ? 0 : 2));
SetThreadAffinityMask( GetCurrentThread(), mask );
}
// Call Intel MKL LAPACK routine
return 0;
}
```

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 supported Intel MKL functions and directly call or inline special optimized small-matrix kernels that bypass error checking. For a list of functions supporting direct call, see [Limitations of the Direct Call](#).

To activate the feature, do the following:

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

Intel MKL Mode	Macro	Compiler Option
Threaded	<code>MKL_DIRECT_CALL</code>	<code>/DMKL_DIRECT_CALL</code>
Sequential	<code>MKL_DIRECT_CALL_SEQ</code>	<code>/DMKL_DIRECT_CALL_SEQ</code>

- 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 call to a function that supports direct call, such as `dgemm`, 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:

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

// Call Intel MKL DGEMM

return 0;
}
```

- 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

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

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

      end
```

- 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 the following functions:
 - BLAS: ?gemm, ?gemm3m, ?syrk, ?trsm, ?axpy, and ?dot
 - LAPACK: ?getrf, ?getrs, and ?getri (available for C applications only)
- 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

This section discusses coding techniques to improve performance on processors based on supported architectures including Intel® Many Integrated Core Architecture. See [Improving Performance on Intel Xeon Phi Coprocessors](#) for tips to improve performance on Intel® Xeon Phi™ coprocessors.

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

Data Alignment and Leading Dimensions

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

For Intel® Xeon Phi™ processor x200 product family, codenamed Knights Landing, align your matrices on 4096-byte boundaries and set the leading dimension to the following integer expression:

$((n * element_size + 511) / 512) * 512 + 64) / element_size$,

where *n* is the matrix dimension along the leading dimension.

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 Developer Reference). Their functionality is strictly equivalent to the functionality of the unpacked routines with the names containing the letters HE, OR, PO, SY, TR, UN in the same positions, but the performance is significantly lower.

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

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

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

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

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

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

See Also

[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 Developer Reference, 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 are using 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:

```
#include "i_malloc.h"
...
i_malloc = my_malloc;
i_calloc = my_calloc;
```

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

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:

```
#include "i_malloc.h"
. . .
i_malloc_dll = my_malloc;
i_calloc_dll = my_calloc;
i_realloc_dll = my_realloc;
i_free_dll   = my_free;
. . .
// Now you may call Intel MKL functions
```

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.

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 Developer Reference 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	
<code>mk1_blas95.lib¹</code>	Fortran 95 wrappers for BLAS (BLAS95) for IA-32 architecture.
<code>mk1_blas95_ilp64.lib¹</code>	Fortran 95 wrappers for BLAS (BLAS95) supporting LP64 interface.
<code>mk1_blas95_lp64.lib¹</code>	Fortran 95 wrappers for BLAS (BLAS95) supporting ILP64 interface.
<code>mk1_lapack95.lib¹</code>	Fortran 95 wrappers for LAPACK (LAPACK95) for IA-32 architecture.
<code>mk1_lapack95_lp64.lib¹</code>	Fortran 95 wrappers for LAPACK (LAPACK95) supporting LP64 interface.

File name	Contains
<code>mk1_lapack95_ilp64.lib¹</code>	Fortran 95 wrappers for LAPACK (LAPACK95) supporting ILP64 interface.
<code>fftw2xc_intel.lib¹</code>	Interfaces for FFTW version 2.x (C interface for Intel compilers) to call Intel MKL FFT.
<code>fftw2xc_ms.lib</code>	Contains interfaces for FFTW version 2.x (C interface for Microsoft compilers) to call Intel MKL FFT.
<code>fftw2xf_intel.lib</code>	Interfaces for FFTW version 2.x (Fortran interface for Intel compilers) to call Intel MKL FFT.
<code>fftw3xc_intel.lib²</code>	Interfaces for FFTW version 3.x (C interface for Intel compiler) to call Intel MKL FFT.
<code>fftw3xc_ms.lib</code>	Interfaces for FFTW version 3.x (C interface for Microsoft compilers) to call Intel MKL FFT.
<code>fftw3xf_intel.lib²</code>	Interfaces for FFTW version 3.x (Fortran interface for Intel compilers) to call Intel MKL FFT.
<code>fftw2x_cdft_SINGLE.lib</code>	Single-precision interfaces for MPI FFTW version 2.x (C interface) to call Intel MKL cluster FFT.
<code>fftw2x_cdft_DOUBLE.lib</code>	Double-precision interfaces for MPI FFTW version 2.x (C interface) to call Intel MKL cluster FFT.
<code>fftw3x_cdft.lib</code>	Interfaces for MPI FFTW version 3.x (C interface) to call Intel MKL cluster FFT.
<code>fftw3x_cdft_ilp64.lib</code>	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	
<code>blas95.mod¹</code>	Fortran 95 interface module for BLAS (BLAS95).
<code>lapack95.mod¹</code>	Fortran 95 interface module for LAPACK (LAPACK95).
<code>f95_precision.mod¹</code>	Fortran 95 definition of precision parameters for BLAS95 and LAPACK95.
<code>mk1_service.mod¹</code>	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 `<mk1_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 `<mk1_directory>\interfaces\blas95` or `<mk1_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:

```
...
#define MKL_STDCALL
#include "mkl.h"
...
```

- 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](#)

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

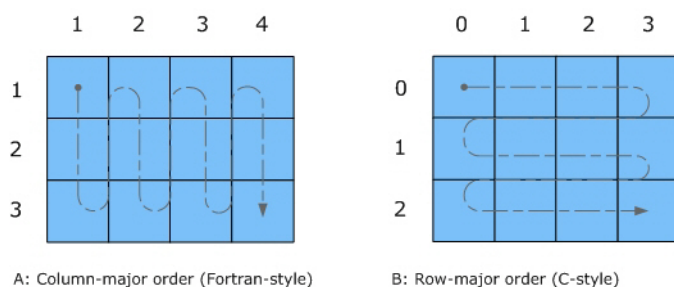
Avoid calling BLAS 95/LAPACK 95 from C/C++. Such calls require skills in manipulating the descriptor of a deferred-shape array, which is the Fortran 90 type. Moreover, BLAS95/LAPACK95 routines contain links to a Fortran RTL.

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).



For example, if a two-dimensional matrix A of size $m \times n$ 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 ($i=0, \dots, m-1, j=0, \dots, n-1$)

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

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 Developer Reference for a description of the C interface to LAPACK functions.

CBLAS

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

CBLAS is a C-style interface to the BLAS routines. You can call CBLAS routines using regular C-style calls. Use the `mk1.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 `mk1.h` header file with the C interface to LAPACK. `mk1.h` includes the `mk1_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 `mk1_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 `mk1_types.h` header file. The only requirement is that the types must be compatible with the Fortran complex layout, that is, the complex type must be a pair of real numbers for the values of real and imaginary parts.

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

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

and

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

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

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

See Also

[Intel® Software Documentation Library](#) for the Intel®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`.

```
#include "mkl.h"
#define N 5
int main()
```

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

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

Below is the C++ implementation:

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

#define N 5

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

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

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

This example uses CBLAS:

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

```
return 0;  
}
```


Obtaining Numerically Reproducible Results

6

Intel® 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.

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

```
mk1_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 Developer Reference*:

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 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 Developer Reference* for how to specify the branches using functions.

See Also

[Getting Started with Conditional Numerical Reproducibility](#)

Reproducibility Conditions

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

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

- 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 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 Developer Reference* for more examples.

C Example of CNR

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

```

}
/* CNR calls to Intel MKL + any other code */
/* Free the allocated aligned array */
mkl_free(darray);
}

```

Fortran Example of CNR

```

PROGRAM MAIN
  INCLUDE 'mkl.fi'
  INTEGER*4 MY_CBWR_BRANCH
! Align all input/output data on 64-byte boundaries
! "for best performance of Intel MKL
! Declare Intel MKL memory allocation routine
#ifdef _IA32
  INTEGER MKL_MALLOC
#else
  INTEGER*8 MKL_MALLOC
#endif
  EXTERNAL MKL_MALLOC, MKL_FREE
  DOUBLE PRECISION DARRAY
  POINTER (P_DARRAY,DARRAY(1))
  INTEGER DARRAY_SIZE
  PARAMETER (DARRAY_SIZE=1000)
! Set alignment value in bytes
  INTEGER ALIGNMENT
  PARAMETER (ALIGNMENT=64)
! Allocate aligned array
  P_DARRAY = MKL_MALLOC (%VAL(8*DARRAY_SIZE), %VAL(ALIGNMENT));
! Find the available MKL_CBWR_BRANCH automatically
  MY_CBWR_BRANCH = MKL_CBWR_GET_AUTO_BRANCH()
! User code without Intel MKL calls
! Piece of the code where CNR of Intel MKL is needed
! The performance of Intel MKL functions may be reduced for CNR mode
! If the "IF" statement below is commented out, Intel MKL will run in a regular mode,
! and data alignment will allow you to get best performance
  IF (MKL_CBWR_SET (MY_CBWR_BRANCH) .NE. MKL_CBWR_SUCCESS) THEN
    PRINT *, 'Error in setting MKL_CBWR_BRANCH! Aborting...'
    RETURN
  ENDIF
! CNR calls to Intel MKL + any other code
! Free the allocated aligned array
  CALL MKL_FREE(P_DARRAY)

END

```

Use of CNR with Unaligned Data in C

```

#include <mkl.h>
int main(void) {
  int my_cbwr_branch;
  /* If it is not possible to align all input/output data on 64-byte boundaries */
  /* to achieve performance, use unaligned IO data with possible performance */
  /* penalty */
  /* Using unaligned IO data */
  double *darray;
  int darray_size=1000;
  /* Allocate array, malloc aligns data on 8/16-byte boundary only */
  darray = (double *)malloc (sizeof(double)*darray_size);
  /* Find the available MKL_CBWR_BRANCH automatically */

```

```

my_cbwr_branch = mkl_cbwr_get_auto_branch();
/* User code without Intel MKL calls */
/* Piece of the code where CNR of Intel MKL is needed */
/* The performance of Intel MKL functions might be reduced for CNR mode */
/* If the "IF" statement below is commented out, Intel MKL will run in a regular mode, */
/* and you will NOT get best performance without data alignment */
if (mkl_cbwr_set(my_cbwr_branch)) {
    printf("Error in setting MKL_CBWR_BRANCH! Aborting...\n");
    return;
}

/* CNR calls to Intel MKL + any other code */
/* Free the allocated array */
free(darray);

```

Use of CNR with Unaligned Data in Fortran

```

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

```


Coding Tips

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

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

```
! ***** Fortran language *****
...
! Set value of alignment
integer    alignment
parameter (alignment=64)
...
! Declare Intel MKL routines
#ifdef _IA32
integer mkl_malloc
#else
integer*8 mkl_malloc
#endif
external mkl_malloc, mkl_free, mkl_app
...
double precision darray
pointer (p_wrk,darray(1))
integer workspace
...
```

```

! Allocate aligned workspace
p_wrk = mkl_malloc( %val(8*workspace), %val(alignment) )
...
! call the program using Intel MKL
call mkl_app( darray )
...
! Free workspace
call mkl_free(p_wrk)

```

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

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

The following preprocessor symbols are available:

Predefined Preprocessor Symbol	Description
<code>__INTEL_MKL__</code>	Intel MKL major version
<code>__INTEL_MKL_MINOR__</code>	Intel MKL minor version
<code>__INTEL_MKL_UPDATE__</code>	Intel MKL update number
<code>INTEL_MKL_VERSION</code>	Intel MKL full version in the following format: <code>INTEL_MKL_VERSION =</code> <code>(__INTEL_MKL__*100+__INTEL_MKL_MINOR__)*100+__I</code> <code>NTEL_MKL_UPDATE__</code>

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

To perform conditional compilation:

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

C/C++ compiler: `mkl_version.h`,
or `mkl.h`, which includes `mkl_version.h`

Intel®Fortran compiler: `mkl.fi`

Any Fortran compiler with enabled preprocessing: `mkl_version.h`
Read the documentation for your compiler for the option that enables preprocessing.

2. [Optionally] Use the following preprocessor directives to check whether the macro is defined:

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

3. Use preprocessor directives for conditional inclusion of code:

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

Example

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

Intel®Fortran Compiler:

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

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

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


Managing Output

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 Developer Reference*, 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: <ul style="list-style-type: none"> • <code>Lnx</code> for Linux* OS • <code>Win</code> for Windows* OS 	

Information	Description	Related Links
	<ul style="list-style-type: none"> OSX for macOS* 	
The host CPU frequency.		
Intel MKL interface layer used by the application.	Possible values: <ul style="list-style-type: none"> stdcall or cdecl on systems based on the IA-32 architecture. lp64 or ilp64 on systems based on the Intel® 64 architecture. 	Using the cdecl and stdcall Interfaces Using the ILP64 Interface vs. LP64 Interface
Intel MKL threading layer used by the application.	Possible values: 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. The number printed is prefixed with NMICDev: . 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 <code>cblas_</code> prefix of CBLAS functions is not printed), you can easily recognize the function by the printed name.	
Values of the arguments.	<ul style="list-style-type: none"> 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. Arrays are printed as addresses (to see the alignment of the data). Integer scalar parameters passed by reference are printed by value. Zero values are printed for <code>NULL</code> references. Character values are printed without quotes. For all parameters passed by reference, the values printed are the values <i>returned by the function</i>. For example, the printed value of the <code>info</code> parameter of a LAPACK function is its value after the function execution. 	

Information	Description	Related Links
Time taken by the function.	<ul style="list-style-type: none"> The time is printed in convenient units (seconds, milliseconds, and so on), which are explicitly indicated. The time may fluctuate from run to run. 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 <code>MKL_CBWR</code> environment variable.	The value printed is prefixed with <code>CNR</code> :	Getting Started with Conditional Numerical Reproducibility
Value of the <code>MKL_DYNAMIC</code> environment variable.	The value printed is prefixed with <code>Dyn</code> :	<code>MKL_DYNAMIC</code>
Status of the Intel MKL memory manager.	The value printed is prefixed with <code>FastMM</code> :	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 <code>TID</code> :	
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 <code>NThr</code> :	Intel MKL-specific Environment Variables for Threading Control
Value of the <code>MKL_HOST_WORKDIVISION</code> environment variable.	The value printed is prefixed with <code>WDiv:HOST</code> :	Automatic Offload Controls
Values of the <code>MKL_MIC_<number>_WORKDIVISION</code> environment variables for each Intel® Xeon Phi™ coprocessor available on the system, where <code><number></code> is the number of the coprocessor.	The first value printed is prefixed with <code>WDiv:<number></code> :	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.

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:

<code><path to MPI binaries></code>	By default, the <code>bin</code> subdirectory in the MPI installation directory.
<code><MPI linker></code>	<code>mpicl</code> or <code>mpiifort</code>

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 (`mk1_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.

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 Developer Reference*.

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 msmapi.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 msmapi.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 msmapi.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 Architecture (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.

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 Developer Reference* 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 Developer Reference* 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	<p>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.</p> <p>In particular, this setting may help you to configure the environment for an MPI application to run Intel MKL in the AO mode.</p> <p>If this variable is not set, all the coprocessors available on the system are used for AO.</p>	<p>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.</p> <p>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.</p>

Environment Variable	Support Function	Description	Value
		<p>You can set this environment variable if AO is enabled by the environment setting or function call.</p> <p>Setting this variable to an empty value is equivalent to completely disabling AO regardless of the value of MKL_MIC_ENABLE.</p> <p>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.</p> <p>For more information, refer to the <i>Intel® Compiler User and Reference Guides</i>.</p>	
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	<code>mkl_mic_set_workdivision</code>	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	<code>mkl_mic_set_workdivision</code>	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	<code>mkl_mic_set_workdivision</code>	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, <i><number></i> 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. Here <i><number></i> 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, <i><number></i> 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. If AO is disabled, this setting has no effect. 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 <i>Intel MKL Release Notes</i>).	Desired behavior of mkl_malloc: 0 - not register allocated memory 1 - register allocated memory

Environment Variable	Support Function	Description	Value
MKL_MIC_RESOURCE_LIMIT	mkl_mic_set_resource_limit	<p>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.</p> <p>Actual reservation is made during a call to an Intel MKL AO function.</p>	<p>A floating-point number ranging from 0.0 to 1.0.</p> <p>Special values:</p> <ul style="list-style-type: none"> 0.0 - do not share Intel Xeon Phi coprocessors across multiple processes. Default. 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 (<i>ppn</i>) from the environment variables passed to the process by MPI. If this attempt is successful, Intel MKL sets the actual resource limit to $1.0/ppn$.

NOTE

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*.

Examples:

- 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.
- Two 58-core Intel Xeon Phi coprocessors are available on the system. The user makes these settings

```
MKL_MIC_1_WORKDIVISION=
0.0
```

```
MKL_MIC_PPN=4
```

```
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. 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. For more information, refer to the <i>Intel® Compiler User and Reference Guides</i> . 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: 0 - No reporting, default. 1 - The report includes: <ul style="list-style-type: none"> The name of the function called in the AO mode. Effective work division. The value of -1 indicates that the hint, that is, the work division specified by the mkl_mic_set_workdivision function or the appropriate MKL*_WORKDIVISION environment variable was ignored in this function call. The time spent on the host CPU during the call. The time spent on each available Intel Xeon Phi coprocessor during the call.

Environment Variable	Support Function	Description	Value
			<p>2 - In addition to the above information, the report includes:</p> <ul style="list-style-type: none"> 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 <code><mkldirectory>/lib/mic</code> .
MKL_MIC_THRESHOLDS_?GEMM	None	Specifies matrix size thresholds for ?GEMM computations in the AO mode.	<p>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.</p> <p>Example: To set the thresholds to $M=2000$, $N=1000$, $K=500$ for DGEMM, set</p> <p><code>MKL_MIC_THRESHOLDS_DGEMM=2000,1000,500</code>.</p>

- 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      Upload A and B to the card, and do not deallocate them after the
c      pragma. C is uploaded and downloaded back, but the allocated memory
c      is retained
c      !DEC$ ATTRIBUTES OFFLOAD : MIC :: SGEMM
c      !DEC$ OFFLOAD TARGET( MIC:0 ) IN( N ), &
c      !DEC$ IN( A: LENGTH( N * N ) ALLOC_IF(.TRUE.) FREE_IF(.FALSE.)), &
c      !DEC$ IN( B: LENGTH( N * N ) ALLOC_IF(.TRUE.) FREE_IF(.FALSE.)), &
c      !DEC$ INOUT( C: LENGTH( N * N ) ALLOC_IF(.TRUE.) FREE_IF(.FALSE.))
c      CALL SGEMM( 'N', 'N', N, N, N, 1.0, A, N, B, N, 1.0, C, N )

c      Change C here

c      Reuse A and B on the card, and upload the new C. Free all the
c      memory on the card
c      !DEC$ ATTRIBUTES OFFLOAD : MIC :: SGEMM
c      !DEC$ OFFLOAD TARGET( MIC:0 ) IN( N ), &
c      !DEC$ NOCOPY( A: LENGTH( N * N ) ALLOC_IF(.FALSE.) FREE_IF(.TRUE.)), &
c      !DEC$ NOCOPY( B: LENGTH( N * N ) ALLOC_IF(.FALSE.) FREE_IF(.TRUE.)), &
c      !DEC$ INOUT( C: LENGTH( N * N ) ALLOC_IF(.FALSE.) FREE_IF(.TRUE.))
c      CALL SGEMM( 'N', 'N', N, N, N, 1.0, A, N, B, N, -1.0, C, N )
```

C

```
/* Upload A and B to the card, and do not deallocate them after the pragma.
 * C is uploaded and downloaded back, but the allocated memory is retained. */
#pragma offload target(mic:0) \
    in(A: length(matrix_elements) alloc_if(1) free_if(0)) \
    in(B: length(matrix_elements) alloc_if(1) free_if(0)) \
    in(transa, transb, N, alpha, beta) \
    inout(C:length(matrix_elements) alloc_if(1) free_if(0))
{
    sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N,
        &beta, C, &N);
}

/* Change C here */

/* Reuse A and B on the card, and upload the new C. Free all the memory on
 * the card. */
#pragma offload target(mic:0) \
    nocopy(A: length(matrix_elements) alloc_if(0) free_if(1)) \
    nocopy(B: length(matrix_elements) alloc_if(0) free_if(1)) \
    in(transa, transb, N, alpha, beta) \
    inout(C:length(matrix_elements) alloc_if(0) free_if(1))
{
    sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N,
        &beta, C, &N);
}
```

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 `<mk1_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.
- `$MKL_PATH` is a user-defined variable that contains `$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$MKL_PATH \
-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 \
$MKL_PATH/libmkl_cdft_core.a \
$MKL_PATH/libmkl_blacs_intelmpi_lp64.a \
$MKL_PATH/libmkl_intel_lp64.a \
$MKL_PATH/libmkl_intel_thread.a \
$MKL_PATH/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.
- *\$MKL_PATH* 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$MKL_PATH \
-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 \
$MKL_PATH/libmkl_cdft_core.a \
$MKL_PATH/libmkl_blacs_intelmpi_lp64.a \
$MKL_PATH/libmkl_intel_lp64.a \
$MKL_PATH/libmkl_intel_thread.a \
$MKL_PATH/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 $4 \times \text{Number-of-Phi-Cores}$ 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 $4 \times (\text{Number-of-Phi-Cores} - 1)$ 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 Developer Reference* 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 <code>mode</code>	Description
Accuracy Control	
<code>VML_HA</code>	high accuracy versions of VM functions
<code>VML_LA</code>	low accuracy versions of VM functions
<code>VML_EP</code>	enhanced performance accuracy versions of VM functions
Denormalized Numbers Handling Control	
<code>VML_FTZDAZ_ON</code>	Faster processing of denormalized inputs is enabled.
<code>VML_FTZDAZ_OFF</code>	Faster processing of denormalized inputs is disabled.
Error Mode Control	
<code>VML_ERRMODE_IGNORE</code>	No action is set for computation errors.
<code>VML_ERRMODE_STDERR</code>	On error, the error text information is written to <code>stderr</code> .
<code>VML_ERRMODE_EXCEPT</code>	On error, an exception is raised.
<code>VML_ERRMODE_CALLBACK</code>	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. 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. 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 <i>Intel MKL Developer Reference</i> for details.
	1	Configures CFFT to merge local FFT calls with local transpositions. 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 <code>DFTI_TRANSPOSE</code> .
<code>enable_soi</code>	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.

CAUTION
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

Instruction Set Specific Dispatching on Intel® Architectures

Intel MKL automatically queries and then dispatches the code path supported on your Intel® processor to the optimal instruction set architecture (ISA) by default. The `MKL_ENABLE_INSTRUCTIONS` environment variable or the `mkl_enable_instructions` support function enables you to dispatch to an ISA-specific code path of your choice. For example, you can run the Intel® Advanced Vector Extensions (Intel® AVX) code path on an Intel processor based on Intel® Advanced Vector Extensions 2 (Intel® AVX2), or you can run the Intel® Streaming SIMD Extensions 4-2 (Intel® SSE4-2) code path on an Intel AVX-enabled Intel processor. This feature is not available on non-Intel processors.

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

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

Value of MKL_ENABLE_INSTRUCTIONS	ISA	Dispatched by Default
AVX512	Intel AVX-512 for systems based on Intel® Xeon® processors	Yes
AVX512_MIC	Intel AVX-512 for systems based on Intel® Xeon Phi™ processors and coprocessors	Yes
AVX2	Intel AVX2	Yes
AVX	Intel AVX	Yes
SSE4_2	Intel SSE4-2	Yes

For more details about the `mkl_enable_instructions` function, including the argument values, see the *Intel MKL Developer Reference*.

For example:

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


```
mkl_enable_instructions(MKL_ENABLE_AVX2)
```
 - Set the environment variable:


```
set MKL_ENABLE_INSTRUCTIONS=AVX2
```

NOTE

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

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

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-fortran-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.

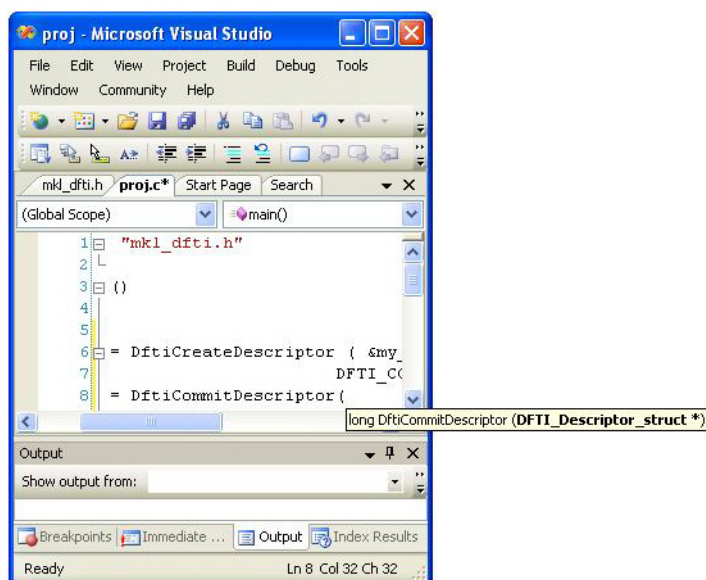
Parameter Info

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:

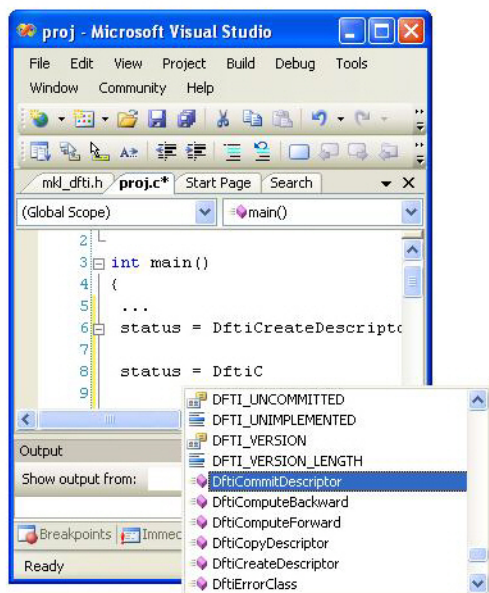


Complete Word

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.



Intel® Math Kernel Library Benchmarks

13

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

Intel® Optimized LINPACK Benchmark for Windows*

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

File in <code>benchmarks\linpack\</code>	Description
<code>linpack_xeon32.exe</code>	The 32-bit program executable for a system based on Intel® Xeon® processor or Intel® Xeon® processor MP with or without Intel® Streaming SIMD Extensions 3 (SSE3).

File in benchmarks \\linpack\\	Description
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.
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 files `lininput_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:

<code>lininput_xeon32</code>	2 GB
<code>lininput_xeon64</code>	16 GB
<code>lininput_xeon64_ao</code>	8 GB
<code>lininput_mic</code>	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*:

- 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>) and

for benchmarking your cluster. To use the benchmark you need to be familiar with HPL usage. The Intel Optimized MP LINPACK Benchmark provides some enhancements designed to make the HPL usage more convenient and to use Intel® Message-Passing Interface (MPI) settings to improve performance.

The Intel Optimized MP LINPACK Benchmark measures the amount of time it takes to factor and solve a random dense system of linear equations ($Ax=b$) in `real*8` precision, converts that time into a performance rate, and tests the results for accuracy. The benchmark uses random number generation and full row pivoting to ensure the accuracy of the results.

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. The prebuilt binaries require Intel® MPI library be installed on the cluster. The run-time version of Intel MPI library is free and can be downloaded from <http://www.intel.com/software/products/>.

NOTE

Intel Optimized MP LINPACK Benchmark prebuilt binaries cannot run with the symmetric model of the Intel MPI library, where the MPI ranks reside on the host and the coprocessors. 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 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® Math Kernel Library (Intel® MKL) license.

Intel MKL provides prebuilt binaries that are linked against Intel MPI libraries either statically or dynamically. In addition, binaries linked with a customized MPI implementation can be created using the Intel MKL MPI wrappers.

NOTE

Performance of statically and dynamically linked prebuilt binaries may be different. The performance of both depends on the version of Intel MPI you are using. You can build binaries statically or dynamically linked against a particular version of Intel MPI by yourself.

HPL code is homogeneous by nature: it requires that each MPI process runs in an environment with similar CPU and memory constraints. The Intel Optimized MP LINPACK Benchmark supports heterogeneity, meaning that the data distribution can be balanced to the performance requirements of each node, provided that there is enough memory on that node to support additional work. For information on how to configure Intel MKL to use the internode heterogeneity, see [Heterogeneous Support in the Intel Optimized MP LINPACK Benchmark](#).

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

The Intel Optimized MP LINPACK Benchmark supports first- and second-generation Intel® Xeon Phi™ coprocessors in offload modes, where the MPI processes are run on the host Intel® Xeon® processor. The second-generation Intel Xeon Phi processor is also supported in native mode, where the MPI processes are run directly on the Intel Xeon Phi processor. The offload mode combines use of different parallelization methods and offloading computations to coprocessors. This mode can be advantageous if the host Intel Xeon processor has more memory than the Intel Xeon Phi coprocessor because the MPI processes have access to more memory when run on the host processors than on the coprocessors. To maximize performance, increase the memory on the host processor or processors (64 GB per coprocessor is ideal) and run a large problem using a large block size. Such runs offload pieces of work to the coprocessors. Although this method increases the PCI Express bus traffic, it is worthwhile for solving a problem that is large enough.

See Also

[Offloading to Intel Xeon Phi Coprocessors](#)

Contents of the Intel Optimized MP LINPACK Benchmark

The Intel Optimized MP LINPACK Benchmark includes prebuilt binaries linked with Intel® MPI library. For a customized MPI implementation, tools are also included to build a binary using Intel MKL MPI wrappers. All the files are located in the `.\benchmarks\mp_linpack\` subdirectory of the Intel MKL directory.

File in <code><mk1_directory>\benchmarks\mp_linpack\</code>	Contents
COPYRIGHT	Original Netlib HPL copyright document.
readme.txt	Information about the files provided.
Prebuilt executables for performance testing	
xhpl_intel64_dynamic.exe	Prebuilt binary for the Intel® 64 architecture dynamically linked against Intel MPI library [‡] . The binary accelerates execution by offloading computations to Intel Xeon Phi coprocessors if they are available on the system.
Run scripts and an input file example	
runme_intel64_dynamic.bat	Sample run script for the Intel® 64 architecture and binary dynamically linked against Intel MPI library.
runme_intel64_prv.bat	Script that sets HPL environment variables. It is called by <code>runme_intel64_dynamic.bat</code> .
HPL.dat	Example of an HPL configuration file.
Prebuilt libraries and utilities for building with a customized MPI implementation	
libhpl_intel64.lib	Library file required to build Intel Optimized MP LINPACK Benchmark for the Intel® 64 architecture with a customized MPI implementation.
HPL_main.c	Source code required to build Intel Optimized MP LINPACK Benchmark for the Intel® 64 architecture with a customized MPI implementation.
build.bat	Build script for creating Intel Optimized MP LINPACK Benchmark for the Intel® 64 architecture with a customized MPI implementation.
Utilities for building Netlib HPL from source code	
Make.Windows_Intel64	Makefile for building Netlib HPL from source code.

[‡] 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](#)

Building the Intel Optimized MP LINPACK Benchmark for a Customized MPI Implementation

The Intel Optimized MP LINPACK Benchmark contains a sample build script `build.bat`. If you are using a customized MPI implementation, this script builds a binary using Intel MKL MPI wrappers. To build the binary, follow these steps:

1. Specify the location of Intel MKL to be used (`MKLROOT`)

2. Set up your MPI environment
3. Run the script `build.bat`

See Also

[Contents of the Intel Optimized MP LINPACK Benchmark](#)

Building the Netlib HPL from Source Code

The source code for Intel Optimized MP LINPACK Benchmark is not provided. However, you can download reference Netlib HPL source code from <http://www.netlib.org/benchmark/hpl/> . To build the HPL:

1. Download and extract the source code.
2. Copy the makefile `<mk1_directory>\benchmarks\mp_linpack\Make.Windows_Intel64` to your HPL directory
3. Edit `Make.Windows_Intel64` as appropriate
4. Build the HPL binary:


```
$> nmake -f Make.Windows_Intel64
```
5. Check that the built binary is available in the current directory.

NOTE

Intel Optimized MP LINPACK Benchmark may contain additional optimizations compared to the reference Netlib HPL implementation.

See Also

[Contents of the Intel Optimized MP LINPACK Benchmark](#)

Configuring Parameters

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

- 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.
 Choose $P \leq Q$.
- NB - the block size of the data distribution.

The table below shows recommended values of NB for different Intel® processors:

Processor	NB
Intel® Xeon® Processor X56*/E56*/E7-*/E7*/X7* (codenamed Nehalem or Westmere)	256
Intel Xeon Processor E26*/E26* v2 (codenamed Sandy Bridge or Ivy Bridge)	256
Intel Xeon Processor E26* v3/E26* v4 (codenamed Haswell or Broadwell)	192
Intel® Core™ i3/i5/i7-6* Processor (codenamed Skylake Client)	192
Intel® Xeon Phi™ Processor 72* (codenamed Knights Landing)	336
Intel Xeon Processor supporting Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instructions (codenamed Skylake Server)	384

For recommended values of NB when offloading to one or more Intel Xeon Phi coprocessors, see [Offloading to Intel Xeon Phi Coprocessors](#).

- 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, see [Heterogeneous Support in the Intel Optimized MP LINPACK Benchmark](#) for how to choose N .

NOTE

Increasing N usually increases performance, but the size of N is bounded by memory. In general, you can compute the memory required to store the matrix (which does not count internal 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`. A general rule of thumb is to choose a problem size that fills 80% of memory. When offloading to Intel Xeon Phi coprocessors, you may choose a problem size that fills 70% of memory, to leave room for additional buffers needed for offloading. Choose N and NB such that $N \gg NB$.

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 file every time you do a new run.

Placeholders in this command line illustrate these parameters:

```
xhpl.exe -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 with the sample `runme` script. For example:

```
runme_intel64_dynamic.bat -m <memory size in Mbytes> -b <block size> -p <grid row dimn> -q <grid column dimn>
```

For more command-line parameters, see [Heterogeneous Support in the Intel Optimized MP LINPACK Benchmark](#).

If you want to run for $N=10000$ on a 1×3 grid, execute this command, provided that the other parameters in `HPL.dat` and the script are correct:

```
runme_intel64_dynamic.bat -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 $NB=256$ on 16 nodes, adjust the script to set the total number of MPI processes to 16 and execute this command:

```
runme_intel64_dynamic.bat -m 50000 -b 256 -p 4 -q 4
```

Running the Intel Optimized MP LINPACK Benchmark

To run the Intel Optimized MP LINPACK Benchmark on multiple nodes or on one node with multiple MPI processes, you need to use MPI and either modify `HPL.dat` or use [Ease-of-use Command-line Parameters](#). The following example describes how to run the dynamically-linked prebuilt Intel Optimized MP LINPACK Benchmark binary using the script provided. To run other binaries, adjust the steps accordingly.

1. Load the necessary environment variables for the Intel MPI Library and Intel® compiler:


```
<parent directory>\bin\compilervars.bat intel64
<mpi directory>\bin64\mpivars.bat
```
2. In `HPL.dat`, set the problem size N to 10000. Because this setting is for a test run, the problem size should be small.
3. For better performance, enable non-uniform memory access (NUMA) on your system and configure to run an MPI process for each NUMA socket as explained below.

NOTE

High-bandwidth Multi-Channel Dynamic Random Access Memory (MCDRAM) on the second-generation Intel Xeon Phi processors may appear to be a NUMA node. However, because there are no CPUs on this node, do not run an MPI process for it.

- Refer to your BIOS settings to enable NUMA on your system.
- Set the following variables at the top of the `runme_intel64_dynamic.bat` script 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.

- In the `HPL.dat` file, set the parameters `Ps` and `Qs` so that $Ps * Qs$ equals the number of MPI processes. For example, for 2 processes, set `Ps` to 1 and `Qs` to 2. Alternatively, leave the `HPL.dat` file as is and launch with `-p` and `-q` command-line parameters.

4. Execute `runme_intel64_dynamic.bat` script:

```
runme_intel64_dynamic.bat
```

5. Rerun the 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 line 6 of `HPL.dat` or use the `-n` command-line parameter:

- For 16 GB: 40000 `Ns`
- For 32 GB: 56000 `Ns`
- For 64 GB: 83000 `Ns`

See Also

Notational Conventions

Building the Intel Optimized MP LINPACK Benchmark for a Customized MPI Implementation

Building the Netlib HPL from Source Code

Offloading to Intel Xeon Phi Coprocessors

Offloading to Intel Xeon Phi Coprocessors

Intel Optimized MP LINPACK Benchmark reacts to `MKL_MIC_ENABLE` and `OFFLOAD_DEVICES` environment variables for automatic offload to Intel Xeon Phi coprocessors. They also inform you how many Intel Xeon Phi coprocessors are detected during the run. The top of the output has a line like this:

```
Number of Intel(R) Xeon Phi(TM) coprocessors: 1
```

This is the number of Intel Xeon Phi coprocessors 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. Specifically, check whether the `LD_LIBRARY_PATH` environment variable contains shared libraries `libcoi_host.so.0` and `libscif.so.0`, which are installed by the Intel® Manycore Platform Software Stack (Intel® MPSS).

You can use environment variables to adjust the behavior of your runs. For a list of supported environment variables, see [Environment Variables](#). The variable `NUMMIC` refers to the number of Intel Xeon Phi coprocessors per cluster node. You can use `HPL_MIC_DEVICE` and `HPL_MIC_SHAREMODE` environment variables to share the Intel Xeon Phi coprocessors among MPI processes. The script `runme_intel64_dynamic.bat` sets these environment variables for you for a given number of MPI ranks per node.

Adjust the block size `NB` and problem size `N` parameters as appropriate. The table below shows recommended values of `NB` for different numbers of Intel Xeon Phi coprocessors per node. The values may vary and depend on the PCI Express settings and performance of main memory.

1 coprocessor	2 coprocessors	3 coprocessors
960	1024	1200

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 Intel Optimized MP LINPACK Benchmark in an offload mode.

Heterogeneous Support in the Intel Optimized MP LINPACK Benchmark

Intel Optimized MP LINPACK Benchmark achieves heterogeneous support by distributing the matrix data unequally between the nodes. The heterogeneous factor command-line parameter f controls the amount of work to be assigned to the more powerful nodes, while the command-line parameter c controls the number of process columns for the faster nodes:

```
xhpl.exe -n <problem size> -b <block size> -p <grid row dimn> -q <grid column dimn> -f
<heterogeneous factor> -c <number of faster processor columns>
```

If the heterogeneous factor is 2.5, roughly 2.5 times the work will be put on the more powerful nodes. The more work you put on the more powerful nodes, the more memory you might be wasting on the other nodes if all nodes have equal amount of memory. If your cluster includes many different types of nodes, you may need multiple heterogeneous factors.

Let P be the number of rows and Q the number of columns in your processor grid ($P \times Q$). The work must be *homogeneous* within each processor column because vertical operations, such as pivoting or panel factorization, 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. The $-f 1.0 -c 0$ setting corresponds 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:

$$N \sim \sqrt{\text{Memory Utilization} * P * Q * \text{Memory Size in Bytes} / 8}$$

Memory Utilization is usually around 0.8 for homogeneous Intel Xeon processor systems. With Intel Xeon Phi coprocessors involved, *Memory Utilization* is around 0.7 due to extra buffers needed for communication. On a heterogeneous system, you might apply a different formula for N for each set of nodes that are the same and then choose the minimum N over all sets. 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 on a homogeneous system $S = P * Q, F = 1$, and $C = Q$. Take N as

$$N \sim \sqrt{\text{Memory Utilization} * P * Q * ((F * P * C) / S) * \text{Memory Size in Bytes} / 8}$$

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 $8N^2 / (P * Q)$ bytes. So a homogeneous run might look like:

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

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 size, 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, 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.exe -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 $C \cdot P$. 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 simply provide `f` and `c` command-line parameters if you need only one heterogeneous factor, you must add lines to the `HPL.dat` input to support multiple heterogeneous factors. For the above example (two processor columns have nodes that are 2.7 times faster), instead of passing `f` and `c` command-line parameters you can modify the `HPL.dat` input file by adding these two lines to the end:

```
1      number of heterogeneous factors
0 1 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).

If instead there are three different types of nodes in a cluster and you need at least 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. 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.

Environment Variables

The table below lists Intel MKL environment variables to control runs of the Intel Optimized MP LINPACK Benchmark.

Environment Variable	Description	Value
<code>HPL_LARGEPAGE</code>	Defines the memory mapping to be used for both the Intel Xeon processor and Intel Xeon Phi coprocessors.	0 or 1: <ul style="list-style-type: none"> 0 - normal memory mapping, default.

Environment Variable	Description	Value
		<ul style="list-style-type: none"> 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.	<p>An integer ranging from 0 to 2:</p> <ul style="list-style-type: none"> 0 - no log is displayed. 1 - only one root node displays a log, exactly the same as the <code>ASYOUGO</code> option provides. 2 - the most detailed log is displayed. All <i>P</i> root nodes in the processor column that owns the current column block display a log.
HPL_HOST_CORE, HPL_HOST_NODE	<p>Specifies cores or Non-Uniform Memory Access (NUMA) nodes to be used.</p> <p>HPL_HOST_NODE requires NUMA mode to be enabled. You can check whether it is enabled by the <code>numactl --hardware</code> command.</p> <p>The default behavior is auto-detection of the core or NUMA node.</p>	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.	A comma-separated list of integers, each ranging from 0 to the largest number of an Intel Xeon Phi coprocessor on the node.
<p>NOTE</p> <p>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.</p>		
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.	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.

Environment Variable	Description	Value
	The default behavior is auto-detection of the number of coprocessors.	
HPL_MIC_CORE, HPL_MIC_NODE	<p>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:</p> <ul style="list-style-type: none"> HPL_MIC_CORE - a specific core. HPL_MIC_NODE - one of the cores on the specified NUMA node. <p>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.</p>	<p>An integer ranging from 0 to the largest number of a core or NUMA node for the coprocessor.</p> <p>Can be provided in a comma-separated list, each integer corresponding to one coprocessor.</p>
HPL_MIC_NUMCORES	Specifies the 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	<p>Specifies whether and how an Intel Xeon Phi coprocessor is shared among two MPI processes.</p> <p>See example 5 for details.</p>	<p>An integer ranging from 0 to 2:</p> <ul style="list-style-type: none"> 0 - no sharing, default 1 - the lower half of the cores will be used for the MPI process. 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.

You can set Intel Optimized MP LINPACK Benchmark 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

#	Settings	Behavior of the Intel Optimized MP Linpack Benchmark
1	Nothing specified	All Intel Xeon processors and all Intel Xeon Phi coprocessors in the cluster are used.

#	Settings	Behavior of the Intel Optimized MP Linpack Benchmark
2	HPL_PNUMMICS=0	Intel Xeon Phi coprocessors are not used.
3	HPL_MIC_DEVICE=0,2 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 Intel Xeon processor cores on NUMA node 1 are used.
5	HPL_MIC_DEVICE=0,1 HPL_MIC_SHAREMODE=0,2	<p>Only Intel Xeon Phi coprocessors 0 and 1 are used:</p> <ul style="list-style-type: none"> On the coprocessor 0, all cores are used. On the coprocessor 1, the upper half of the cores is used. <p>For a 61-core Intel Xeon Phi coprocessor, the upper half includes cores 31-61.</p> <p>This setting is useful to share an Intel Xeon Phi coprocessor among two MPI processes for an odd number of Intel Xeon Phi coprocessors.</p>

Improving Performance of Your Cluster

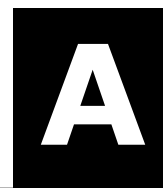
To improve cluster performance, follow these steps, provided all required software is installed on each node:

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 Intel Optimized MP LINPACK Benchmark on every node. Ensure results are within 10% 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, such as one found in the Intel® MPI Benchmarks package.
4. Run an Intel Optimized MP LINPACK Benchmark on pairs of two or four nodes and ensure results are within 10% of each other. The problem size must be large enough depending on the memory size and CPU speed.
5. Run a small problem size over the complete cluster to ensure correctness.
6. Increase the problem size and run the real test load.
7. In case of problems go back to step 2.

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

Intel® Math Kernel Library

Language Interfaces Support



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/ 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
Deep Neural Network (DNN) functions		Yes

Function Domain	Fortran interface	C/C++ interface
Extended Eigensolver	Yes	Yes
Support functions (including memory allocation)	Yes	Yes

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

Include Files

The table below lists Intel MKL include files.

Function Domain/ Purpose	Fortran Include Files	C/C++ Include Files
All function domains	<code>mkl.fi</code>	<code>mkl.h</code>
BLACS		<code>mkl_blacs.h^{††}</code>
BLAS	<code>blas.f90</code> <code>mkl_blas.fi[†]</code>	<code>mkl_blas.h[†]</code>
BLAS-like Extension Transposition Routines	<code>mkl_trans.fi[†]</code>	<code>mkl_trans.h[†]</code>
CBLAS Interface to BLAS		<code>mkl_cblas.h[†]</code>
Sparse BLAS	<code>mkl_spgblas.fi[†]</code>	<code>mkl_spgblas.h[†]</code>
LAPACK	<code>lapack.f90</code> <code>mkl_lapack.fi[†]</code>	<code>mkl_lapack.h[†]</code>
C Interface to LAPACK		<code>mkl_lapacke.h[†]</code>
PBLAS		<code>mkl_pblas.h^{††}</code>
ScaLAPACK		<code>mkl_scalapack.h^{††}</code>
Intel MKL PARDISO	<code>mkl_pardiso.f90</code> <code>mkl_pardiso.fi[†]</code>	<code>mkl_pardiso.h[†]</code>
Parallel Direct Sparse Solvers for Clusters	<code>mkl_cluster_ sparse_solver.f90</code>	<code>mkl_cluster_ sparse_solver.h[†]</code>
Direct Sparse Solver (DSS)	<code>mkl_dss.f90</code> <code>mkl_dss.fi[†]</code>	<code>mkl_dss.h[†]</code>
RCI Iterative Solvers		
ILU Factorization	<code>mkl_rci.f90</code> <code>mkl_rci.fi[†]</code>	<code>mkl_rci.h[†]</code>
Optimization Solver	<code>mkl_rci.f90</code> <code>mkl_rci.fi[†]</code>	<code>mkl_rci.h[†]</code>
Vector Mathematics	<code>mkl_vml.90</code> <code>mkl_vml.fi[†]</code>	<code>mkl_vml.h[†]</code>
Vector Statistics	<code>mkl_vsl.f90</code> <code>mkl_vsl.fi[†]</code>	<code>mkl_vsl.h[†]</code>
Fast Fourier Transforms	<code>mkl_dfti.f90</code>	<code>mkl_dfti.h[†]</code>

Function Domain/ Purpose	Fortran Include Files	C/C++ Include Files
Cluster Fast Fourier Transforms	<code>mkl_cdft.f90</code>	<code>mkl_cdft.h^{##}</code>
Partial Differential Equations Support		
Trigonometric Transforms	<code>mkl_trig_transforms.f90</code>	<code>mkl_trig_transform.h[†]</code>
Poisson Solvers	<code>mkl_poisson.f90</code>	<code>mkl_poisson.h[†]</code>
Data Fitting	<code>mkl_df.f90</code>	<code>mkl_df.h[†]</code>
Deep Neural Networks		<code>mkl_dnn.h[†]</code>
Extended Eigensolver	<code>mkl_solvers_ee.fi[†]</code>	<code>mkl_solvers_ee.h[†]</code>
Support functions	<code>mkl_service.f90</code> <code>mkl_service.fi[†]</code>	<code>mkl_service.h[†]</code>
Declarations for replacing memory allocation functions. See Redefining Memory Functions for details.		<code>i_malloc.h</code>
Auxiliary macros to determine the version of Intel MKL at compile time.	<code>mkl_version</code>	<code>mkl_version[†]</code>

[†] 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

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 Developer Reference 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

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.

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		
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			
mkl_intel_thread.lib	OpenMP threading library for the Intel compilers		

File	Contents	Optional Component	
		Name	Installed by Default
mkl_tbb_thread.lib	Intel® Threading Building Blocks (Intel® TBB) threading library for the Intel compilers	Intel TBB threading support	Yes
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		
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
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\mk1` 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
<code>mk1_rt.dll</code>	Single Dynamic Library		
Threading Layer			
<code>mk1_intel_thread.dll</code>	Dynamic OpenMP threading library for the Intel compilers		
<code>mk1_tbb_thread.dll</code>	Dynamic Intel TBB threading library for the Intel compilers	Intel TBB threading support	Yes
<code>mk1_sequential.dll</code>	Dynamic sequential library		
Computational Layer			
<code>mk1_core.dll</code>	Core library containing processor-independent code and a dispatcher for dynamic loading of processor-specific code		
<code>mk1_p4.dll</code>	Pentium® 4 processor kernel		
<code>mk1_p4m.dll</code>	Kernel library for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) enabled processors		
<code>mk1_p4m3.dll</code>	Kernel library for Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) enabled processors		
<code>mk1_avx.dll</code>	Kernel library for Intel® Advanced Vector Extensions (Intel® AVX) enabled processors		
<code>mk1_avx2.dll</code>	Kernel library for Intel® Advanced Vector Extensions 2 (Intel® AVX2) enabled processors		

File	Contents	Optional Component	
		Name	Installed by Default
mk1_avx512.dll	Kernel library for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) enabled processors		
mk1_vml_p4.dll	Vector Mathematics (VM)/Vector Statistics (VS)/Data Fitting (DF) part of Pentium® 4 processor kernel		
mk1_vml_p4m.dll	VM/VS/DF for Intel® SSSE3 enabled processors		
mk1_vml_p4m2.dll	VM/VS/DF for 45nm Hi-k Intel® Core™2 and Intel Xeon® processor families		
mk1_vml_p4m3.dll	VM/VS/DF for Intel® SSE4.2 enabled processors		
mk1_vml_avx.dll	VM/VS/DF optimized for Intel® AVX enabled processors		
mk1_vml_avx2.dll	VM/VS/DF optimized for Intel® AVX2 enabled processors		
mk1_vml_avx512.dll	VM/VS/DF optimized for Intel® AVX-512 enabled processors		
mk1_vml_ia.dll	VM/VS/DF default kernel for newer Intel® architecture processors		
libmk1_vml_cmpt.dll	VM/VS/DF library for conditional numerical reproducibility		
libimalloc.dll	Dynamic library to support renaming of memory functions		
Message Catalogs			
1033\mk1_msg.dll	Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English		
1041\mk1_msg.dll	Catalog of Intel MKL messages in Japanese. Available only if Intel		

File	Contents	Optional Component	
		Name	Installed by Default
	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			
mk1_intel_lp64.lib	LP64 interface library for the Intel compilers		
mk1_intel_ilp64.lib	ILP64 interface library for the Intel compilers		
mk1_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
mk1_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
mk1_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
mk1_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			

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* 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	
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
<code>mkl_rt.lib</code>	Single Dynamic Library to be used for linking		
Interface Layer			
<code>mkl_intel_lp64_dll.lib</code>	LP64 interface library for dynamic linking with the Intel compilers		
<code>mkl_intel_ilp64_dll.lib</code>	ILP64 interface library for dynamic linking with the Intel compilers		
Threading Layer			
<code>mkl_intel_thread_dll.lib</code>	OpenMP threading library for dynamic linking with the Intel compilers		
<code>mkl_tbb_thread_dll.lib</code>	Intel TBB threading library for the Intel compilers	Intel TBB threading support	Yes
<code>mkl_pgi_thread_dll.lib</code>	OpenMP threading library for dynamic linking with the PGI* compiler	PGI* Compiler support	
<code>mkl_sequential_dll.lib</code>	Sequential library for dynamic linking		
Computational Layer			
<code>mkl_core_dll.lib</code>	Core library for dynamic linking		
Cluster Libraries			
<code>mkl_scalapack_lp64_dll.lib</code>	ScaLAPACK routine library for dynamic linking supporting the LP64 interface	Cluster support	
<code>mkl_scalapack_ilp64_dll.lib</code>	ScaLAPACK routine library for dynamic linking supporting the ILP64 interface	Cluster support	
<code>mkl_cdft_core_dll.lib</code>	Cluster FFT library for dynamic linking	Cluster support	

File	Contents	Optional Component	
		Name	Installed by Default
mk1_blacs_lp64_dll.lib	LP64 version of BLACS interface library for dynamic linking	Cluster support	
mk1_blacs_ilp64_dll.lib	ILP64 version of BLACS interface library for dynamic linking	Cluster support	

Contents of the `redist\intel64_win\mk1` 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
mk1_rt.dll	Single Dynamic Library		
Threading layer			
mk1_intel_thread.dll	Dynamic OpenMP threading library for the Intel compilers		
mk1_tbb_thread.dll	Dynamic Intel TBB threading library for the Intel compilers	Intel TBB threading support	Yes
mk1_pgi_thread.dll	Dynamic OpenMP threading library for the PGI* compiler	PGI* compiler support	
mk1_sequential.dll	Dynamic sequential library		
Computational layer			
mk1_core.dll	Core library containing processor-independent code and a dispatcher for dynamic loading of processor-specific code		
mk1_def.dll	Default kernel for the Intel® 64 architecture		
mk1_mc.dll	Kernel library for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) enabled processors		

File	Contents	Optional Component	
		Name	Installed by Default
mk1_mc3.dll	Kernel library for Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) enabled processors		
mk1_avx.dll	Kernel library optimized for Intel® Advanced Vector Extensions (Intel® AVX) enabled processors		
mk1_avx2.dll	Kernel library optimized for Intel® Advanced Vector Extensions 2 (Intel® AVX2) enabled processors		
mk1_avx512.dll	Kernel library optimized for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) enabled processors		
mk1_vml_def.dll	Vector Mathematics (VM)/Vector Statistics (VS)/Data Fitting (DF) part of default kernel		
mk1_vml_mc.dll	VM/VS/DF for Intel® SSSE3 enabled processors		
mk1_vml_mc2.dll	VM/VS/DF for 45nm Hi-k Intel® Core™2 and Intel Xeon® processor families		
mk1_vml_mc3.dll	VM/VS/DF for Intel® SSE4.2 enabled processors		
mk1_vml_avx.dll	VM/VS/DF optimized for Intel® AVX enabled processors		
mk1_vml_avx2.dll	VM/VS/DF optimized for Intel® AVX2 enabled processors		
mk1_vml_avx512.dll	VM/VS/DF optimized for Intel® AVX-512 enabled processors		
libmk1_vml_cmpt.dll	VM/VS/DF library for conditional numerical reproducibility		

File	Contents	Optional Component	
		Name	Installed by Default
libimalloc.dll	Dynamic library to support renaming of memory functions		
Cluster Libraries			
mk1_scalapack_lp64.dll	ScaLAPACK routine library supporting the LP64 interface	Cluster support	
mk1_scalapack_ilp64.dll	ScaLAPACK routine library supporting the ILP64 interface	Cluster support	
mk1_cdft_core.dll	Cluster FFT dynamic library	Cluster support	
mk1_blacs_lp64.dll	LP64 version of BLACS routines	Cluster support	
mk1_blacs_ilp64.dll	ILP64 version of BLACS routines	Cluster support	
mk1_blacs_intelmpi_lp64.dll	LP64 version of BLACS routines for Intel® MPI Library	Cluster support	
mk1_blacs_intelmpi_ilp64.dll	ILP64 version of BLACS routines for Intel MPI Library	Cluster support	
mk1_blacs_mpich2_lp64.dll	LP64 version of BLACS routines for MPICH2 or higher	Cluster support	
mk1_blacs_mpich2_ilp64.dll	ILP64 version of BLACS routines for MPICH2 or higher	Cluster support	
mk1_blacs_msmpi_lp64.dll	LP64 version of BLACS routines for Microsoft* MPI	Cluster support	
mk1_blacs_msmpi_ilp64.dll	ILP64 version of BLACS routines for Microsoft* MPI	Cluster support	
Message Catalogs			
1033\mk1_msg.dll	Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English		
1041\mk1_msg.dll	Catalog of Intel MKL messages in Japanese. Available only if Intel MKL provides Japanese		

File	Contents	Optional Component	
		Name	Installed by Default
	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
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

File	Contents
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

A

affinity mask62
aligning data, example87
architecture support27
Automatic Offload Mode, concept104

B

BLAS
 calling routines from C73
 Fortran 95 interface to70
 OpenMP* threaded routines49
building a custom DLL
 in Visual Studio* IDE47

C

C interface to LAPACK, use of73
C, calling LAPACK, BLAS, CBLAS from73
C/C++, Intel(R) MKL complex types74
calling
 BLAS functions from C75
 CBLAS interface from C75
 complex BLAS Level 1 function from C75
 complex BLAS Level 1 function from C++75
 Fortran-style routines from C73
calling convention, cdecl and stdcall22
CBLAS interface, use of73
cdecl interface, use of38
Cluster FFT
 environment variable for120
 linking with96
 linking with on Intel® Xeon Phi™ coprocessors113
 managing performance of120
cluster software, Intel(R) MKL95
cluster software, linking with
 commands96
 linking examples100
 on Intel® Xeon Phi™ coprocessors113
Cluster Sparse Solver, linking with96
code examples, use of23
coding
 data alignment87
 techniques to improve performance65
compilation, Intel(R) MKL version-dependent88
Compiler Assisted Offload
 examples111
compiler run-time libraries, linking with43
compiler support22
compiler-dependent function71
complex types in C and C++, Intel(R) MKL74
computation results, consistency79
computational libraries, linking with42
computations offload to Intel(R) Xeon Phi(TM)
 coprocessors
 automatic104
 compiler assisted, examples111
 interoperability of offload techniques113
 techniques103
conditional compilation88
configuring
 Intel(R) Visual Fortran124

 Microsoft Visual* C/C++123
consistent results79
context-sensitive Help, for Intel(R) MKL, in Visual Studio*
 IDE124
conventions, notational17
ctdcall interface, use of38
custom DLL
 building44
 composing list of functions46
 specifying function names46
CVF calling convention, use with Intel(R) MKL72

D

data alignment, example87
denormal number, performance66
direct call, to Intel(R) Math Kernel Library computational
 kernels63
directory structure
 high-level27
 in-detail
dispatch Intel(R) architectures, configure with an
 environment variable121
dispatch, new Intel(R) architectures, enable with an
 environment variable121

E

Enter index keyword31
environment variables
 for threading control56
 setting for specific architecture and programming
 interface21
 to control dispatching for Intel(R) architectures121
 to control threading algorithm for ?gemm59
 to enable dispatching of new architectures121
 to manage behavior of function domains119
 to manage behavior of Intel(R) Math Kernel Library
 with119
 to manage performance of cluster FFT120
examples, linking
 for cluster software100
 for Intel(R) Xeon Phi(TM) coprocessors112
general34

F

FFT interface
 OpenMP* threaded problems49
 optimised radices66
FFTW interface support145
Fortran 95 interface libraries41
function call information, enable printing91

H

header files, Intel(R) MKL142
heterogeneity
 of Intel(R) Optimized MP LINPACK Benchmark129
heterogeneous cluster
 support by Intel(R) Optimized MP LINPACK
 Benchmark for Clusters135
HT technology, configuration tip62

I

- ILP64 programming, support for39
- improve performance, for matrices of small sizes63
- include files, Intel(R) MKL142
- information, for function call , enable printing91
- installation, checking21
- Intel(R) Hyper-Threading Technology, configuration tip62
- Intel(R) Many Integrated Core Architecture
 - examples of linking for coprocessors, for Compiler-Assisted Offload112
 - improving performance on116
 - Intel(R) MKL usage on103
- Intel(R) Optimized MP LINPACK Benchmark for Clusters
 - heterogeneity of129
 - heterogeneous support135
 - usage modes for Intel® Xeon Phi™ coprocessors130
- Intel(R) Visual* Fortran project, linking with Intel(R) MKL32
- Intel(R) Xeon Phi(TM) coprocessors
 - Intel(R) MKL usage on103
 - linking for112
 - performance tips for116
 - running Intel(R) Math Kernel Library in native mode on113
- Intel® Many Integrated Core Architecture
 - usage of Intel(R) Optimized MP LINPACK Benchmark for Clusters for130
- Intel® Threading Building Blocks, functions threaded with51
- IntelliSense*, with Intel(R) MKL, in Visual Studio* IDE124
- interface
 - cdecl and stdcall, use of38
 - Fortran 95, libraries41
 - LP64 and ILP64, use of39
- interface libraries and modules, Intel(R) MKL69
- interface libraries, linking with38

K

- kernel, in Intel(R) Math Kernel Library, direct call to63

L

- language interfaces support141
- language-specific interfaces
 - interface libraries and modules69
- LAPACK
 - C interface to, use of73
 - calling routines from C73
 - Fortran 95 interface to70
 - OpenMP* threaded routines49
 - performance of packed routines65
- layers, Intel(R) MKL structure29
- libraries to link with
 - computational42
 - interface38
 - run-time43
 - system libraries44
 - threading41
- link tool, command line34
- linking
 - for Intel(R) Xeon Phi(TM) coprocessors112
 - Intel(R) Visual* Fortran project with Intel(R) MKL32
 - Microsoft Visual* C/C++ project with Intel(R) MKL32
- linking examples
 - cluster software100

- general34
- linking with
 - compiler run-time libraries43
 - computational libraries42
 - interface libraries38
 - system libraries44
 - threading libraries41
- linking, quick start31
- linking, Web-based advisor34
- LINPACK benchmark127

M

- memory functions, redefining67
- memory management67
- memory renaming67
- message-passing interface
 - custom, usage99
 - Intel(R) Math Kernel Library interaction with99
 - support95
- Microsoft Visual* C/C++ project, linking with Intel(R) MKL32
- mixed-language programming73
- module, Fortran 9570
- MPI
 - custom, usage99
 - Intel(R) Math Kernel Library interaction with99
 - support95
- multi-core performance62

N

- native run, of Intel(R) Math Kernel Library on Intel(R) Xeon Phi(TM) coprocessors113
- notational conventions17
- number of OpenMP threads
 - for Intel(R) Math Kernel Library on Intel(R) Xeon Phi(TM) coprocessors116
- number of threads
 - changing at run time53
 - changing with OpenMP* environment variable53
 - Intel(R) MKL choice, particular cases57
 - setting for cluster97
 - techniques to set52
- numerically reproducible results79

O

- offload techniques, interoperability113
- OpenMP* threaded functions49
- OpenMP* threaded problems49

P

- parallel performance52
- parallelism, of Intel(R) MKL49
- performance
 - multi-core62
 - on Intel(R) Many Integrated Core Architecture116
 - with denormals66
 - with subnormals66
- performance improvement, for matrices of small sizes63
- performance, of Intel(R) MKL, improve on specific processors66

R

results, consistent, obtaining79
results, numerically reproducible, obtaining79

S

ScaLAPACK, linking with
 on Intel® Xeon Phi™ coprocessors113
SDL33, 36
Single Dynamic Library33, 36
stdcall calling convention, use in C/C++71
structure
 high-level27
 in-detail
 model29
support, technical11
supported architectures27
system libraries, linking with44

T

technical support11

thread safety, of Intel(R) MKL49
threaded functions, with Intel® Threading Building
 Blocks51
threading control, Intel(R) MKL-specific56
threading libraries, linking with41

U

unstable output, getting rid of79

V

Vector Mathematics
 default mode, setting with environment variable119
 environment variable to set default mode119
verbose mode, of Intel(R) MKL91
Visual Studio* IDE
 IntelliSense*, with Intel(R) MKL124
 using Intel(R) MKL context-sensitive Help in124

