



Intel® Math Kernel Library for macOS*

Developer Guide

Intel® MKL 2017 -macOS*

Revision: 054

Legal Information

Contents

Legal Information.....	7
Getting Help and Support.....	9
Introducing the Intel® Math Kernel Library.....	11
What's New.....	13
Notational Conventions.....	15
Related Information.....	17
Chapter 1: Getting Started	
Checking Your Installation.....	19
Setting Environment Variables	19
Compiler Support.....	20
Using Code Examples.....	21
What You Need to Know Before You Begin Using the Intel® Math Kernel Library.....	21
Chapter 2: Structure of the Intel® Math Kernel Library	
Architecture Support.....	23
High-level Directory Structure.....	23
Layered Model Concept.....	24
Chapter 3: Linking Your Application with the Intel® Math Kernel Library	
Linking Quick Start.....	27
Using the -mkl Compiler Option.....	27
Using the Single Dynamic Library.....	28
Selecting Libraries to Link with.....	28
Using the Link-line Advisor.....	29
Using the Command-line Link Tool.....	29
Linking Examples.....	30
Linking on IA-32 Architecture Systems.....	30
Linking on Intel(R) 64 Architecture Systems.....	31
Linking in Detail.....	32
Listing Libraries on a Link Line.....	32
Dynamically Selecting the Interface and Threading Layer.....	33
Linking with Interface Libraries.....	34
Using the ILP64 Interface vs. LP64 Interface.....	34
Linking with Fortran 95 Interface Libraries.....	36
Linking with Threading Libraries.....	36
Linking with Compiler Run-time Libraries.....	38
Linking with System Libraries.....	38
Building Custom Dynamically Linked Shared Libraries	38
Using the Custom Dynamically Linked Shared Library Builder.....	38
Composing a List of Functions	39
Specifying Function Names.....	40
Distributing Your Custom Dynamically Linked Shared Library.....	41
Chapter 4: Managing Performance and Memory	

Improving Performance with Threading.....	43
OpenMP* Threaded Functions and Problems.....	43
Functions Threaded with Intel® Threading Building Blocks.....	45
Avoiding Conflicts in the Execution Environment.....	46
Techniques to Set the Number of Threads.....	46
Setting the Number of Threads Using an OpenMP* Environment Variable.....	47
Changing the Number of OpenMP* Threads at Run Time.....	47
Using Additional Threading Control.....	49
Intel MKL-specific Environment Variables for OpenMP Threading Control.....	50
MKL_DYNAMIC.....	51
MKL_DOMAIN_NUM_THREADS.....	51
MKL_NUM_STRIPES.....	53
Setting the Environment Variables for Threading Control.....	54
Calling Intel MKL Functions from Multi-threaded Applications.....	54
Using Intel® Hyper-Threading Technology.....	55
Improving Performance for Small Size Problems	56
Using MKL_DIRECT_CALL in C Applications.....	56
Using MKL_DIRECT_CALL in Fortran Applications.....	57
Limitations of the Direct Call	57
Other Tips and Techniques to Improve Performance.....	58
Coding Techniques.....	58
Improving Intel(R) MKL Performance on Specific Processors.....	59
Operating on Denormals.....	59
Fast Fourier Transform Optimized Radices.....	59
Using Memory Functions	59
Avoiding Memory Leaks in Intel MKL.....	59
Redefining Memory Functions.....	60

Chapter 5: Language-specific Usage Options

Using Language-Specific Interfaces with Intel® Math Kernel Library.....	61
Interface Libraries and Modules.....	61
Fortran 95 Interfaces to LAPACK and BLAS.....	62
Compiler-dependent Functions and Fortran 90 Modules.....	63
Mixed-language Programming with the Intel Math Kernel Library.....	63
Calling LAPACK, BLAS, and CBLAS Routines from C/C++ Language Environments.....	64
Using Complex Types in C/C++.....	65
Calling BLAS Functions that Return the Complex Values in C/C++ Code..	66

Chapter 6: Obtaining Numerically Reproducible Results

Getting Started with Conditional Numerical Reproducibility	70
Specifying Code Branches.....	71
Reproducibility Conditions.....	72
Setting the Environment Variable for Conditional Numerical Reproducibility.....	73
Code Examples.....	73

Chapter 7: Coding Tips

Example of Data Alignment.....	77
--------------------------------	----

Using Predefined Preprocessor Symbols for Intel® MKL Version-Dependent Compilation.....	78
Chapter 8: Managing Output	
Using Intel MKL Verbose Mode.....	81
Version Information Line.....	81
Call Description Line.....	82
Chapter 9: Working with the Intel® Math Kernel Library Cluster Software	
Linking with Intel MKL Cluster Software.....	85
Setting the Number of OpenMP* Threads.....	86
Using Shared Libraries.....	87
Interaction with the Message-passing Interface.....	87
Using a Custom Message-Passing Interface.....	88
Examples of Linking for Clusters.....	89
Examples for Linking a C Application.....	89
Examples for Linking a Fortran Application.....	90
Chapter 10: Managing Behavior of the Intel(R) Math Kernel Library with Environment Variables	
Managing Behavior of Function Domains.....	91
Setting the Default Mode of Vector Math with an Environment Variable...	91
Managing Performance of the Cluster Fourier Transform Functions.....	92
Instruction Set Specific Dispatching on Intel® Architectures.....	93
Chapter 11: Configuring Your Integrated Development Environment to Link with Intel(R) Math Kernel Library	
Configuring the Apple Xcode* Developer Software to Link with Intel® Math Kernel Library.....	95
Chapter 12: Intel® Math Kernel Library Benchmarks	
Intel® Optimized LINPACK Benchmark for macOS*.....	97
Contents of the Intel® Optimized LINPACK Benchmark.....	97
Running the Software.....	98
Known Limitations of the Intel® Optimized LINPACK Benchmark.....	99
Appendix A: Intel® Math Kernel Library Language Interfaces Support	
Language Interfaces Support, by Function Domain.....	101
Include Files.....	102
Appendix B: Support for Third-Party Interfaces	
FFTW Interface Support.....	105
Appendix C: Directory Structure in Detail	
Static Libraries in the lib directory.....	107
Dynamic Libraries in the lib directory.....	109

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.

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

© 2017, Intel Corporation.

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

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/>.

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.

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-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Optimization Notice

Notice revision #20110804

What's New

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

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

- 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](#).
- The names of Apple* operating systems have been updated.

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.

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

The following notations are used to refer to Intel 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,
`icc myprog.c -L$MKLPATH -I$MKLINCLUDE -lmkl -liomp5 -lpthread`
- Filenames, directory names, and pathnames, for example,
`/System/Library/Frameworks/JavaVM.framework/Versions/1.5/Home`
- C/C++ code fragments, for example,
`a = new double [SIZE*SIZE];`

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

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

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

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

Related Information

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

- The *Intel® 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 OS X* 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.
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 following files appear in the `<mkl directory>/bin` directory:
`mklvars.sh`
`mklvars.csh`
Use these files 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 macOS* is complete, set the `INCLUDE`, `MKLROOT`, `DYLD_LIBRARY_PATH`, `MANPATH`, `LIBRARY_PATH`, `CPATH`, and `NLSPATH` environment variables in the command shell using one of the script files in the `bin` subdirectory of the Intel MKL installation directory.

Choose the script corresponding to your command shell:

S Script File

h
e
l
l

C mklvars.csh

B mklvars.sh

a
s
h

The scripts accept the parameters, explained in the following table:

Setting Specified	Required (Yes/No)	Possible Values	Comment
Architecture	Yes, when applicable	ia32 intel64	
Use of Intel MKL Fortran modules precompiled with the Intel® Fortran compiler	No	mod	Supply this parameter only if you are using this compiler.
Programming interface (LP64 or ILP64)	No	lp64, default ilp64	

For example:

- The command `mklvars.sh ia32` sets the environment for Intel MKL to use the IA-32 architecture.
- The command `mklvars.sh intel64 mod ilp64` sets the environment for Intel MKL to use the Intel 64 architecture, ILP64 programming interface, and Fortran modules.
- The command `mklvars.sh intel64 mod` sets the environment for Intel MKL to use the Intel 64 architecture, LP64 interface, and Fortran modules.

NOTE

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

See Also

[High-level Directory Structure](#)

[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.

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

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

[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. Source code for the examples is in the next-level `sources` subdirectory.

See Also

[High-level Directory Structure](#)

What You Need to Know Before You Begin Using the Intel® 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: 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 line syntax and libraries for static and dynamic linking are different. For the list of link libraries for static and dynamic models, linking examples, and other relevant topics, like how to save disk space by creating a custom dynamic library, see [Linking Your Application with the Intel® Math Kernel Library](#).

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 macOS* supports the IA-32, Intel® 64, and compatible architectures in its universal libraries, located in the `<mkl_directory>/lib` directory.

NOTE

Universal libraries contain both 32-bit and 64-bit code. If these libraries are used for linking, the linker dispatches appropriate code as follows:

- A 32-bit linker dispatches 32-bit code and creates 32-bit executable files.
- A 64-bit linker dispatches 64-bit code and creates 64-bit executable files.

See Also

[High-level Directory Structure](#)

[Notational Conventions](#)

[Directory Structure in Detail](#)

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>	Scripts to set environmental variables in the user shell
<code>bin/ia32</code>	Shell scripts for the IA-32 architecture
<code>bin/intel64</code>	Shell scripts for the Intel® 64 architecture
<code>benchmarks/linpack</code>	Shared-Memory (SMP) version of LINPACK benchmark
<code>examples</code>	Source and data files for Intel MKL examples. Provided in archives corresponding to Intel MKL components selected during installation.

Directory	Contents
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/fftw	Header files for the FFTW2 and FFTW3 interfaces
interfaces/blas95	Fortran 95 interfaces to BLAS and a makefile to build the library
interfaces/fftw2xc	FFTW 2.x interfaces to Intel MKL FFT (C interface)
interfaces/fftw2xf	FFTW 2.x interfaces to Intel MKL FFT (Fortran interface)
interfaces/fftw3xc	FFTW 3.x interfaces to Intel MKL FFT (C interface)
interfaces/fftw3xf	FFTW 3.x interfaces to Intel MKL FFT (Fortran interface)
interfaces/lapack95	Fortran 95 interfaces to LAPACK and a makefile to build the library
lib	Universal static libraries and shared objects for the IA-32 and Intel® 64 architectures
tools	Tools and plug-ins
tools/builder	Tools for creating custom dynamically linkable libraries

See Also

[Notational Conventions](#)

[Using Code Examples](#)

Layered Model Concept

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

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

You can combine Intel MKL libraries to meet your needs by linking with one library in each part layer-by-layer.

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

The following table provides more details of each layer.

Layer	Description
Interface Layer	This layer matches compiled code of your application with the threading and/or computational parts of the library. This layer provides: <ul style="list-style-type: none"> • LP64 and ILP64 interfaces.

Layer	Description
Threading Layer	<ul style="list-style-type: none"><li data-bbox="430 222 1271 249">• Compatibility with compilers that return function values differently. <p data-bbox="430 285 553 312">This layer:</p> <ul style="list-style-type: none"><li data-bbox="430 333 1287 361">• Provides a way to link threaded Intel MKL with supported compilers.<li data-bbox="430 365 1300 392">• Enables you to link with a threaded or sequential mode of the library. <p data-bbox="430 413 1365 468">This layer is compiled for different environments (threaded or sequential) and compilers (from Intel and PGI*).</p>
Computational Layer	This layer accommodates multiple architectures through identification of architecture features and chooses the appropriate binary code at run time.

See Also[Using the ILP64 Interface vs. LP64 Interface](#)[Linking Your Application with the Intel® 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, which depend on the way you link:

Using the Intel® Parallel Studio XE Composer Edition compiler	see Using the -mkl Compiler Option .
Explicit dynamic linking	see Using the Single Dynamic Library for how to simplify your link line.
Explicitly listing libraries on your link line	see Selecting Libraries to Link with for a summary of the libraries.
Using 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 -mkl Compiler Option

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

`-mkl` or
`-mkl=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

`-mkl=sequential`

to link with sequential version of Intel MKL.

`-mkl=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 `-mkl` compiler option, see the Intel Compiler User and Reference Guides.

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

If you specify any variant of the `-mkl` 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

[Listing Libraries on a Link Line](#)

[Using the ILP64 Interface vs. LP64 Interface](#)

[Using the Link-line Advisor](#)

[Intel® Software Documentation Library](#) for Intel® compiler documentation

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 `libmkl_rt.dylib` on your link line. For example:

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

Here `MKLPATH=$MKLROOT/lib`.

SDL enables you to select the interface and threading library for Intel 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>libmkl_intel.a</code>	<code>libmkl_intel_thread.a</code>	<code>libmkl_core.a</code>	<code>libiomp5.dylib</code>

	Interface layer	Threading layer	Computational layer	RTL
IA-32 architecture, dynamic linking	libmkl_intel.dylib	libmkl_intel_thread.dylib	libmkl_core.dylib	libiomp5.dylib
Intel® 64 architecture, static linking	libmkl_intel_lp64.a	libmkl_intel_thread.a	libmkl_core.a	libiomp5.dylib
Intel® 64 architecture, dynamic linking	libmkl_intel_lp64.dylib	libmkl_intel_thread.dylib	libmkl_core.dylib	libiomp5.dylib

The Single Dynamic Library (SDL) automatically links interface, threading, and computational libraries and thus simplifies linking. The following table lists Intel 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	libmkl_rt.dylib	libiomp5.dylib [†]

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

For exceptions and alternatives to the libraries listed above, see [Linking in Detail](#).

See Also

[Layered Model Concept](#)

[Using the Link-line Advisor](#)

[Using the -mkl Compiler Option](#)

Using the Link-line Advisor

Use the Intel 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` 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

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 `icc`.

In these examples,

`MKLPATH=$MKLROOT/lib,`

`MKLINCLUDE=$MKLROOT/include.`

NOTE

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

- Static linking of `myprog.f` and OpenMP* threaded Intel MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
$MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
$MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
-liomp5 -lpthread -lm
```

- Dynamic linking of `myprog.f` and OpenMP* threaded Intel MKL:

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

- Static linking of `myprog.f` and sequential version of Intel MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
$MKLPATH/libmkl_intel.a $MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a
$MKLPATH/libmkl_intel.a $MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a
-lpthread -lm
```

- Dynamic linking of `myprog.f` and sequential version of Intel MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -Wl,-rpath,$MKLPATH
-lmkl_intel -lmkl_sequential -lmkl_core -lpthread -lm
```

- Dynamic linking of `myprog.f` and OpenMP* threaded or sequential Intel MKL (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 -L$MKLPATH -Wl,-rpath,$MKLPATH -lmkl_rt
```

- Static linking of `myprog.f`, Fortran 95 LAPACK interface, and OpenMP* threaded Intel MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/ia32
-lmkl_lapack95
$MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
```

- Static linking of `myprog.f`, Fortran 95 BLAS interface, and OpenMP* threaded Intel MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/ia32
-lmkl_blas95 $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/
libmkl_core.a $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/
libmkl_core.a -liomp5 -lpthread -lm
```

- Static linking of `myprog.c` and Intel MKL threaded with Intel® Threading Building Blocks (Intel® TBB), provided that the `LIBRARY_PATH` environment variable contains the path to Intel TBB library:

```
icc myprog.c -I$MKLINCLUDE -Wl,--start-group $MKLPATH/libmkl_intel.a $MKLPATH/
libmkl_tbb_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group -ltbb -lstdc++
-lpthread -lm
```

- Dynamic linking of `myprog.c` and Intel MKL threaded with Intel TBB, provided that the `LIBRARY_PATH` environment variable contains the path to Intel TBB library:

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

See Also

[Fortran 95 Interfaces to LAPACK and BLAS](#)

[Linking with System Libraries](#) for specifics of linking with a GNU compiler

Linking on Intel(R) 64 Architecture Systems

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

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

In these examples,

`MKLPATH=$MKLROOT/lib,`

`MKLINCLUDE=$MKLROOT/include.`

NOTE

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

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
$MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
```

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -Wl,-rpath,$MKLPATH
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
-liomp5 -lpthread -lm
```

- Static linking of `myprog.f` and sequential version of Intel MKL supporting the LP64 interface:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
$MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_sequential.a
$MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_sequential.a
$MKLPATH/libmkl_core.a -lpthread -lm
```

- Dynamic linking of `myprog.f` and sequential version of Intel MKL supporting the LP64 interface:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -Wl,-rpath,$MKLPATH
-lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread -lm
```

- Static linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the ILP64 interface:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
$MKLPATH/libmkl_intel_ilp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_ilp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
```

- Dynamic linking of `myprog.f` and OpenMP* threaded Intel MKL supporting the ILP64 interface:

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

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

```
ifort myprog.f -L$MKLPATH -Wl,-rpath,$MKLPATH -lmkl_rt
```

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/intel64/lp64
-lmkl_lapack95_lp64 $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
```

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/intel64/lp64
-lmkl_blas95_lp64 $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -liomp5 -lpthread -lm
```

- Static linking of `myprog.c` and Intel MKL threaded with Intel® Threading Building Blocks (Intel® TBB), provided that the `LIBRARY_PATH` environment variable contains the path to Intel TBB library:

```
icc myprog.c -L$MKLPATH -I$MKLINCLUDE -Wl,--start-group $MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_tbb_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group -ltbb -lstdc++
-lpthread -lm
```

- Dynamic linking of `myprog.c` and Intel MKL threaded with Intel TBB, provided that the `LIBRARY_PATH` environment variable contains the path to Intel TBB library:

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

See Also

[Fortran 95 Interfaces to LAPACK and BLAS](#)

[Linking with System Libraries](#) for specifics of linking with a GNU or PGI compiler

Linking in Detail

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

Listing Libraries on a Link Line

To link with Intel MKL, specify paths and libraries on the link line as shown below.

NOTE

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

<files to link>

`-L<MKL path>-I<MKL include>`

`[-I<MKL include>/{ia32|intel64|{ilp64|lp64}}]`

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


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

-lmkl_{intel|intel_ilp64|intel_lp64}
-lmkl_{intel_thread|tbb_thread|sequential}
-lmkl_core

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

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

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

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

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

See Also

[Using the Link-line Advisor](#)

[Linking Examples](#)

Dynamically Selecting the Interface and Threading Layer

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

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>
Intel TBB threading	TBB	<code>MKL_THREADING_TBB</code>

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.

See Also

- [Using the Single Dynamic Library Layered Model Concept](#)
- [Directory Structure in Detail](#)

Linking with Interface Libraries

Using the ILP64 Interface vs. LP64 Interface

The Intel 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:

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

The ILP64 interface provides for the following:

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

The LP64 interface provides compatibility with the previous Intel 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	<code>ifort -i8 -I<mkl directory>/include ...</code>

Fortran

 Compiling for LP64 `ifort -I<mk1 directory>/include ...`

C or C++

 Compiling for ILP64 `icc -DMKL_ILP64 -I<mk1 directory>/include ...`

 Compiling for LP64 `icc -I<mk1 directory>/include ...`

CAUTION

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

Coding for ILP64

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

To migrate to ILP64 or write new code for ILP64, use appropriate types for parameters of the Intel 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 `libmkl_blas95*.a` and `libmkl_lapack95*.a` 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 `libiomp5.dylib` library, located under `<parent directory>/compiler/lib`. This RTL is compatible with the GNU* compilers (`gcc` and `gfortran`). You can find additional information about the Intel OpenMP RTL at <https://www.openmpRTL.org>.

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

In the *sequential mode*, Intel MKL runs unthreaded code, does not require a 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: libmkl_intel_thread.a Dynamic linking: libmkl_intel_thread.dylib	libiomp5.dylib
no	any compiler	parallel	Static linking: libmkl_tbb_thread.a Dynamic linking: libmkl_tbb_thread.dylib	libtbb.dylib
no	any compiler	sequential	Static linking: libmkl_sequential.a Dynamic linking: libmkl_sequential.dylib	none [†]
yes	Intel compiler	parallel	Static linking: libmkl_intel_thread.a Dynamic linking: libmkl_intel_thread.dylib	libiomp5.dylib
yes	GNU compiler	parallel	Static linking: libmkl_intel_thread.a Dynamic linking: libmkl_intel_thread.dylib	libiomp5.dylib
yes	PGI* compiler	parallel	Static linking: libmkl_pgi_thread.a Dynamic linking: libmkl_pgi_thread.dylib	PGI OpenMP RTL
yes	any other compiler	parallel	Not supported. Use Intel MKL in the sequential mode.	

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

See Also

[Layered Model Concept](#)
[Notational Conventions](#)

Linking with Compiler Run-time Libraries

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

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

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

See Also

[Setting Environment Variables](#)
[Layered Model Concept](#)

Linking with System Libraries

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

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

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

NOTE

To link with Intel MKL statically using a GNU or PGI compiler, link also the system library `libdl` by adding `-ldl` to your link line. The Intel compiler always passes `-ldl` to the linker.

See Also

[Linking Examples](#)

Building Custom Dynamically Linked Shared Libraries

Custom dynamically linked shared libraries 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 dynamically linked shared library builder enables you to create a dynamic ally linked shared library containing the selected functions and located in the `tools/builder` directory. The builder contains a makefile and a definition file with the list of functions.

Using the Custom Dynamically Linked Shared Library Builder

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

```
make target [<options>]
```

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

Value	Comment
libuni	The builder uses static Intel MKL interface, threading, and core libraries to build a universal dynamically linked shared library for the IA-32 or Intel® 64 architecture.
dylibuni	The builder uses the single dynamic library <code>libmkl_rt.dylib</code> to build a universal dynamically linked shared library for the IA-32 or Intel® 64 architecture.
help	The command prints Help on the custom dynamically linked shared library builder

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

Parameter [Values]	Description
<code>interface = {lp64 ilp64}</code>	Defines whether to use LP64 or ILP64 programming interface for the Intel 64 architecture. 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> .
<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 shared object. The default name is <code>user_example_list</code> (no extension).
<code>name = <so name></code>	Specifies the name of the library to be created. By default, the names of the created library is <code>mkl_custom.dylib</code> .
<code>xerbla = <error handler></code>	Specifies the name of the object file <code><user_xerbla>.o</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.
<code>MKLROOT = <mkl directory></code>	Specifies the location of Intel MKL libraries used to build the custom dynamically linked shared library. By default, the builder uses the Intel MKL installation directory.

All the above parameters are optional.

In the simplest case, the command line is `make ia32`, and the missing options have default values. This command creates the `mkl_custom.dylib` library for processors using the IA-32 architecture. The command takes the list of functions from the `user_list` file and uses the native Intel MKL error handler `xerbla`.

An example of a more complex case follows:

```
make ia32 export=my_func_list.txt name=mkl_small xerbla=my_xerbla.o
```

In this case, the command creates the `mkl_small.dylib` library 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.o`.

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

See Also

[Using the Single Dynamic Library](#)

Composing a List of Functions

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

1. Link your application with installed Intel 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 dynamically linked shared library, adjust function names to the required interface. For example, for Fortran functions append an underscore character "_" to the names as a suffix:

```

dgemm_
ddot_
dgetrf_

```

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

NOTE

The lists of functions are provided in the `<mkl_directory>/tools/builder` folder merely as examples. See [Composing a List of Functions](#) for how to compose lists of functions for your custom dynamically linked shared library.

TIP

Names of Fortran-style routines (BLAS, LAPACK, etc.) can be both upper-case or lower-case, with or without the trailing underscore. For example, these names are equivalent:

```

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

```

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

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

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

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

For the names of the Fortran support functions, see the [tip](#).

NOTE

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

Distributing Your Custom Dynamically Linked Shared Library

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

Managing Performance and Memory

4

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 `?lacon`) work correctly during simultaneous execution by multiple threads. In particular, any chunk of threaded Intel MKL code provides access for multiple threads to the same shared data, while permitting only one thread at any given time to access a shared piece of data. Therefore, you can call Intel MKL from multiple threads and not worry about the function instances interfering with each other.

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

By default, Intel MKL uses the number of OpenMP threads equal to the number of physical cores on the system.

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

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

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, ?tptrs, ?tbtrs
- Orthogonal factorization, computational routines:
?geqrf, ?ormqr, ?unmqr, ?ormlq, ?unmlq, ?ormql, ?unmq1, ?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, cggevz/zggevz, 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*).
- Sparse BLAS.
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: <code>pthread</code> s on macOS*).	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>libmkl_sequential.a</code> or <code>libmkl_sequential.dylib</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>libmkl_sequential.a</code> or <code>libmkl_sequential.dylib</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).

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:

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

See Also

[Using Additional Threading Control](#)

Changing the Number of OpenMP* Threads at Run Time

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

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

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

```

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

```



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

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

Using Additional Threading Control

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
<code>MKL_NUM_THREADS</code>	<code>mkl_set_num_threads</code> <code>mkl_set_num_threads_local</code>	Suggests the number of OpenMP threads to use.	<code>OMP_NUM_THREADS</code>
<code>MKL_DOMAIN_NUM_THREADS</code>	<code>mkl_domain_set_num_threads</code>	Suggests the number of OpenMP threads for a particular function domain.	
<code>MKL_DYNAMIC</code>	<code>mkl_set_dynamic</code>	Enables Intel MKL to dynamically change the number of OpenMP threads.	<code>OMP_DYNAMIC</code>

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:

<code>MKL_DOMAIN_ALL</code>	All function domains
<code>MKL_DOMAIN_BLAS</code>	BLAS Routines
<code>MKL_DOMAIN_FFT</code>	Fourier Transform Functions
<code>MKL_DOMAIN_VML</code>	Vector Mathematics (VM)
<code>MKL_DOMAIN_PARDISO</code>	Intel MKL PARDISO, a direct sparse solver based on Parallel Direct Sparse Solver (PARDISO*)

For example,

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

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

The table below illustrates how values of `MKL_DOMAIN_NUM_THREADS` are interpreted.

Value of <code>MKL_DOMAIN_NUM_THREADS</code>	Interpretation
<code>MKL_DOMAIN_ALL=4</code>	All parts of Intel MKL should try four OpenMP threads. The actual number of threads may be still different because of the <code>MKL_DYNAMIC</code> setting or system resource issues. The setting is equivalent to <code>MKL_NUM_THREADS = 4</code> .
<code>MKL_DOMAIN_ALL=1, MKL_DOMAIN_BLAS=4</code>	All parts of Intel MKL should try one OpenMP thread, except for BLAS, which is suggested to try four threads.
<code>MKL_DOMAIN_VML=2</code>	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 );
```

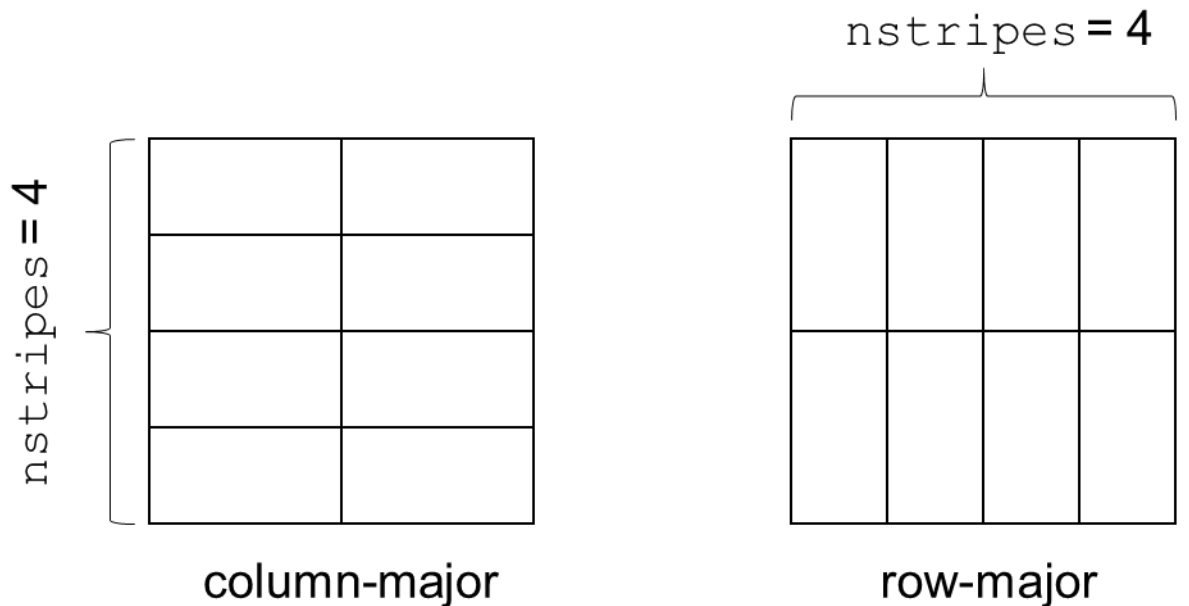
MKL_NUM_STRIPES

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 MKL_NUM_STRIPES	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:

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

For example:

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

Calling 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, POSIX* 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.dylib` 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.dylib` and select a threading library using an environment variable or a function call:
 - set `MKL_THREADING_LAYER=intel` or `MKL_THREADING_LAYER=tbb`
 - call `mk1_set_threading_layer(MKL_THREADING_INTEL)` or `mk1_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.

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:

```
icc -DMKL_DIRECT_CALL -std=c99 your_application.c $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
```



```
$(MKLROOT)/lib/intel64/libmkl_core.a
$(MKLROOT)/lib/intel64/libmkl_intel_thread.a -lpthread -lm -openmp -I$(MKLROOT)/include
```

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

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

Using MKL_DIRECT_CALL in Fortran Applications

The following examples of code and link lines show how to activate direct calls to Intel 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 $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a $(MKLROOT)/lib/intel64/libmkl_intel_thread.a
-lpthread -lm -openmp -I$(MKLROOT)/include
```

- To use Intel 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 $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a $(MKLROOT)/lib/intel64/libmkl_sequential.a
-lpthread -lm -I$(MKLROOT)/include
```

Limitations of the Direct Call

Directly calling the Intel 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 `-std=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

Coding Techniques

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

Data Alignment and Leading Dimensions

To improve performance of your application that calls Intel 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](#).

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

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:

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
```

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	
libmkl_blas95.a ¹	Fortran 95 wrappers for BLAS (BLAS95) for IA-32 architecture.
libmkl_blas95_ilp64.a ¹	Fortran 95 wrappers for BLAS (BLAS95) supporting LP64 interface.
libmkl_blas95_lp64.a ¹	Fortran 95 wrappers for BLAS (BLAS95) supporting ILP64 interface.
libmkl_lapack95.a ¹	Fortran 95 wrappers for LAPACK (LAPACK95) for IA-32 architecture.
libmkl_lapack95_lp64.a ¹	Fortran 95 wrappers for LAPACK (LAPACK95) supporting LP64 interface.

File name	Contains
<code>libmkl_lapack95_ilp64.a</code> ¹	Fortran 95 wrappers for LAPACK (LAPACK95) supporting ILP64 interface.
<code>libfftw2xc_intel.a</code> ¹	Interfaces for FFTW version 2.x (C interface for Intel compilers) to call Intel MKL FFT.
<code>libfftw2xc_gnu.a</code>	Interfaces for FFTW version 2.x (C interface for GNU compilers) to call Intel MKL FFT.
<code>libfftw2xf_intel.a</code>	Interfaces for FFTW version 2.x (Fortran interface for Intel compilers) to call Intel MKL FFT.
<code>libfftw2xf_gnu.a</code>	Interfaces for FFTW version 2.x (Fortran interface for GNU compiler) to call Intel MKL FFT.
<code>libfftw3xc_intel.a</code> ²	Interfaces for FFTW version 3.x (C interface for Intel compiler) to call Intel MKL FFT.
<code>libfftw3xc_gnu.a</code>	Interfaces for FFTW version 3.x (C interface for GNU compilers) to call Intel MKL FFT.
<code>libfftw3xf_intel.a</code> ²	Interfaces for FFTW version 3.x (Fortran interface for Intel compilers) to call Intel MKL FFT.
<code>libfftw3xf_gnu.a</code>	Interfaces for FFTW version 3.x (Fortran interface for GNU compilers) to call Intel MKL FFT.

Modules, in architecture- and interface-specific subdirectories of the Intel MKL include directory

<code>blas95.mod</code> ¹	Fortran 95 interface module for BLAS (BLAS95).
<code>lapack95.mod</code> ¹	Fortran 95 interface module for LAPACK (LAPACK95).
<code>f95_precision.mod</code> ¹	Fortran 95 definition of precision parameters for BLAS95 and LAPACK95.
<code>mkl_service.mod</code> ¹	Fortran 95 interface module for Intel MKL support functions.

¹ Prebuilt for the Intel® Fortran compiler

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

See Also

[Fortran 95 Interfaces to LAPACK and BLAS](#)

Fortran 95 Interfaces to LAPACK and BLAS

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

1. Go to the respective directory `<mkl directory>/interfaces/blas95` or `<mkl directory>/interfaces/lapack95`
2. Type one of the following commands depending on your architecture:
 - For the IA-32 architecture,

```
make libia32 INSTALL_DIR=<user dir>
```
 - For the Intel® 64 architecture,

```
make libintel64 [interface=lp64|ilp64] INSTALL_DIR=<user dir>
```

Important

The parameter `INSTALL_DIR` is required.

As a result, the required library is built and installed in the `<user dir>/lib` directory, and the `.mod` files are built and installed in the `<user dir>/include/<arch>[/{lp64|ilp64}]` directory, where `<arch>` is one of `{ia32, intel64}`.

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

```
FC=<compiler>.
```

For example, the command

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

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

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

- For the IA-32 architecture,

```
make cleania32 INSTALL_DIR=<user dir>
```

- For the Intel® 64 architecture,

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

- For all the architectures,

```
make clean INSTALL_DIR=<user dir>
```

CAUTION

Even if you have administrative rights, avoid setting `INSTALL_DIR=../..` or `INSTALL_DIR=<mk1 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.

Mixed-language Programming with the Intel Math Kernel Library

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

See also these Knowledge Base articles:

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

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

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

CAUTION

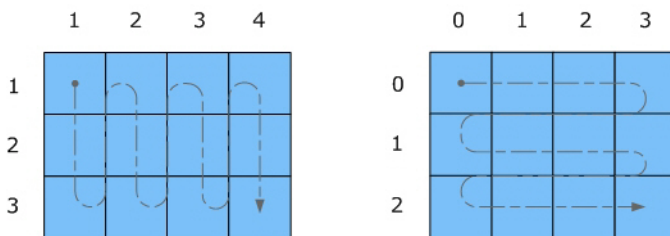
Avoid calling BLAS 95/LAPACK 95 from C/C++. Such calls require skills in manipulating the descriptor of a deferred-shape array, which is the Fortran 90 type. Moreover, BLAS95/LAPACK95 routines contain links to a Fortran RTL.

LAPACK and BLAS

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

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

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



A: Column-major order (Fortran-style)

B: Row-major order (C-style)

For example, if a two-dimensional matrix A of size $m \times n$ 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® 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® 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:

```
export MKL_CBWR = AVX2
```

NOTE

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

Intel CPUs supporting Intel SSE2

To ensure Intel 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:

```
export MKL_CBWR = SSE2
```

NOTE

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

Intel or Intel compatible CPUs supporting Intel SSE2

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

To ensure Intel MKL calls return the same results on all Intel or Intel compatible CPUs supporting Intel SSE2 instructions:

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

- Call

```
mkl_cbwr_set(MKL_CBWR_COMPATIBLE)
```

- Set the environment variable:

```
export MKL_CBWR = COMPATIBLE
```

NOTE

The special `MKL_CBWR_COMPATIBLE/COMPATIBLE` option is provided because Intel and Intel compatible CPUs have a few instructions, such as approximation instructions `rcpps/rsqrtps`, that may return different results. This option ensures that Intel 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 in each list sets Intel MKL to run in the CNR mode based on the default dispatching for your platform. The other two commands in each list are equivalent and set the CNR branch to Intel AVX.

For the bash shell:

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

For the C shell (csh or tcsh):

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

See Also

[Specifying Code Branches](#)

Code Examples

The following simple programs show how to obtain reproducible results from run to run of Intel 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: $INTEL_MKL_VERSION = (_INTEL_MKL_ _ * 100 + _INTEL_MKL_MINOR_ _) * 100 + _INTEL_MKL_UPDATE_ _$

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

To perform conditional compilation:

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

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

Intel®Fortran compiler: `mkl.fi`

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

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

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

3. Use preprocessor directives for conditional inclusion of code:

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

Example

This example shows how to compile a code segment conditionally for a specific version of Intel 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
#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"> No value on systems based on the IA-32 architecture. lp64 or ilp64 on systems based on the Intel® 64 architecture. 	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 following is an example of a version information line:

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

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. 	
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. 	

Information	Description	Related Links
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> :	MKL_DYNAMIC
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

The following is an example of a call description line:

```
MKL_VERBOSE DGEMM(n,n,
1000,1000,240,0x7ffff708bb30,0x7ff2aea4c000,1000,0x7ff28e92b000,240,0x7ffff708bb38,0x7ff28e08d000,1000) 1.66ms CNR:OFF Dyn:1 FastMM:1 TID:0 NThr:16,FFT:2
```

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 architecture.

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

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

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

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

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

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

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

<MKL cluster library>

One of libraries for ScaLAPACK or Cluster FFT and appropriate architecture and programming interface (LP64 or ILP64). Available libraries are listed in [Appendix C: Directory Structure](#)

in Detail. For example, for the LP64 interface, it is -
`lmkl_scalapack_lp64` or `-lmkl_cdft_core`. Cluster Sparse Solver does not require an additional computation library.

<BLACS>

The BLACS library corresponding to your architecture, programming interface (LP64 or ILP64), and MPI used. Available BLACS libraries are listed in [Appendix C: Directory Structure in Detail](#). Specifically, choose one of -
`lmkl_blacs_intelmpi_lp64` or -
`lmkl_blacs_intelmpi_ilp64`.

<MKL core libraries>

Processor optimized kernels, threading library, and system library for threading support, linked as described in [Listing Libraries on a Link Line](#).

<MPI linker script>

A linker script that corresponds to the MPI version.

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

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

TIP

Use the [Using the Link-line Advisor](#) to quickly choose the appropriate set of <MKL cluster Library>, <BLACS>, and <MKL core libraries>.

See Also

[Linking Your Application with the Intel® Math Kernel Library Examples of Linking for Clusters](#)

Setting the Number of OpenMP* Threads

The OpenMP* run-time library responds to the environment variable `OMP_NUM_THREADS`. Intel 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 (`libmkl_intel_thread.a`), this value is the number of CPUs according to the OS.

CAUTION

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

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

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

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

Important

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

See Also

[Techniques to Set the Number of Threads](#)

Using Shared Libraries

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

If Intel MKL is installed only on one node, link statically when building your Intel MKL applications rather than use shared libraries.

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

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

[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 `make` command.

For example: the command

```
make libintel64
```

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

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

```
make help
```

Using a Custom BLACS Library

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

See Also

[Linking with Intel 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 instance, the script `mpirun` is used in the case of MPICH, and the number of MPI processes is set by `-np`. In the case of MPICH, start the daemon before running your application; the execution is driven by the script `mpiexec`.

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

See Also

[Directory Structure in Detail](#)

Examples for Linking a C Application

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

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

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

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

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

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

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

See Also

[Linking with Intel MKL Cluster Software](#)

Using the Link-line Advisor

Examples for Linking a Fortran Application

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

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

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

```

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

```

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

```

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

```

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

```

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

```

See Also

[Linking with Intel MKL Cluster Software](#)
[Using the Link-line Advisor](#)

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

10

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 in your command shell:

- For the bash shell:

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

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

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

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

Values of the `mode` Parameter

Value of <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.

Value of <i>mode</i>	Description
VML_ERRMODE_STDERR	On error, the error text information is written to <i>stderr</i> .
VML_ERRMODE_EXCEPT	On error, an exception is raised.
VML_ERRMODE_CALLBACK	On error, an additional error handler function is called.
VML_ERRMODE_DEFAULT	On error, an exception is raised and an additional error handler function is called.

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

- For the bash shell:

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

- For a C shell (csh or tcsh):

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

NOTE

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

Managing Performance of the Cluster Fourier Transform Functions

Performance of Intel 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 <code>MPI_Alltoallv</code> function to perform global transpositions.
	1	Configures CFFT to use a series of calls to <code>MPI_Isend</code> and <code>MPI_Irecv</code> instead of the <code>MPI_Alltoallv</code> function.
	4	Configures CFFT to merge global transposition with data movements in the local memory. CFFT performs global transpositions by calling <code>MPI_Isend</code> and <code>MPI_Irecv</code> 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.

Option	Possible Values	Description
	1	Configures CFFT to merge local FFT calls with local transpositions. CFFT usually performs faster with this value than with <code>wo_omatcopy = 1</code> if the configuration parameter <code>DFTI_TRANSPOSE</code> has the value of <code>DFTI_ALLOW</code> . See the <i>Intel MKL Developer Reference</i> for details.
	-1 (default)	Enables CFFT to decide which of the two above values to use depending on the value of <code>DFTI_TRANSPOSE</code> . CFFT usually performs faster with this value than with <code>wo_omatcopy = 0</code> if <code>DFTI_TRANSPOSE</code> has the value of <code>DFTI_NONE</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:

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

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 <code>MKL_ENABLE_INSTRUCTIONS</code>	ISA	Dispatched by Default
AVX512	Intel AVX-512 for systems based on Intel® Xeon® processors	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:
 - For the bash shell:


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


```
setenv MKL_ENABLE_INSTRUCTIONS AVX2
```

NOTE

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

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

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

11

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

This section provides information on configuring the Apple Xcode* developer software for linking with Intel MKL.

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

To configure your Xcode developer software, do the following:

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

See Also

[Notational Conventions](#)

[Linking in Detail](#)

Intel® Math Kernel Library Benchmarks

12

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 macOS*

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

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

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

Additional information on this software, as well as on other Intel® software performance products, is available at <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 macOS* 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_cd32</code>	The 32-bit program executable for a system using Intel® Core™ microarchitecture on macOS.
<code>linpack_cd64</code>	The 64-bit program executable for a system using Intel® Core™ microarchitecture on macOS.

File in <code>./benchmarks/ linpack/</code>	Description
<code>runme32</code>	A sample shell script for executing a pre-determined problem set for <code>linpack_cd32</code> .
<code>runme64</code>	A sample shell script for executing a pre-determined problem set for <code>linpack_cd64</code> .
<code>lininput</code>	Input file for a pre-determined problem for the <code>runme32</code> script.
<code>help.lpk</code>	Simple help file.
<code>xhelp.lpk</code>	Extended help file.
These files are not available immediately after installation and appear as a result of execution of an appropriate <code>runme</code> script.	
<code>lin_cd32.txt</code>	Result of the <code>runme32</code> script execution.
<code>lin_cd64.txt</code>	Result of the <code>runme64</code> 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:

```
./runme32
./runme64
```

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_cd32 -e
./linpack_cd64 -e
```

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

`lininput` requires at least 2 GB of memory.

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

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

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

Optimization Notice

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 macOS*:

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

Intel® 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_spbblas.fi[†]</code>	<code>mkl_spbblas.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		<code>mkl_rci.h[†]</code>
ILU Factorization	<code>mkl_rci.f90</code> <code>mkl_rci.fi[†]</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_vs1.f90</code> <code>mkl_vs1.fi[†]</code>	<code>mkl_vs1.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 `<mk1 directory>/lib` directory.

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

Static Libraries in the `lib` directory

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

File	Contents	Optional Component	
		Name	Installed by Default
Interface Layer			
<code>libmk1_intel.a</code>	Interface library for the Intel compilers. To be used on IA-32 architecture systems		
<code>libmk1_intel_lp64.a</code>	Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support LP64 interface		
<code>libmk1_intel_ilp64.a</code>	Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support ILP64 interface		
<code>libmk1_blas95.a</code>	Fortran 95 interface library for BLAS for the Intel® Fortran compiler. To be used on IA-32 architecture systems.	Fortran 95 interfaces for BLAS and LAPACK	Yes
<code>libmk1_blas95_lp64.a</code>	Fortran 95 interface library for BLAS for the Intel® Fortran compiler.	Fortran 95 interfaces for BLAS and LAPACK	Yes

File	Contents	Optional Component	
		Name	Installed by Default
	To be used on Intel® 64 architecture systems to support LP64 interface.		
libmkl_blas95_ilp64.a	Fortran 95 interface library for BLAS for the Intel® Fortran compiler. To be used on Intel® 64 architecture systems to support ILP64 interface.	Fortran 95 interfaces for BLAS and LAPACK	Yes
libmkl_lapack95.a	Fortran 95 interface library for LAPACK for the Intel Fortran compiler. To be used on IA-32 architecture systems.	Fortran 95 interfaces for BLAS and LAPACK	Yes
libmkl_lapack95_lp64.a	Fortran 95 interface library for LAPACK for the Intel® Fortran compiler. To be used on Intel® 64 architecture systems to support LP64 interface.	Fortran 95 interfaces for BLAS and LAPACK	Yes
libmkl_lapack95_ilp64.a	Fortran 95 interface library for LAPACK for the Intel® Fortran compiler. To be used on Intel® 64 architecture systems to support ILP64 interface.	Fortran 95 interfaces for BLAS and LAPACK	Yes
Threading Layer			
libmkl_intel_thread.a	OpenMP threading library for the Intel compilers		
libmkl_tbb_thread.a	Intel® Threading Building Blocks (Intel® TBB) threading library for the Intel compilers	Intel TBB threading support	Yes
libmkl_pgi_thread.a	OpenMP threading library for the PGI* compiler	PGI* compiler support	
libmkl_sequential.a	Sequential library		
Computational Layer			
libmkl_core.a	Kernel library		
Cluster Libraries			

File	Contents	Optional Component	
		Name	Installed by Default
libmkl_scalapack_lp64.a	ScaLAPACK routine library supporting the LP64 interface	Cluster support	
libmkl_scalapack_ilp64.a	ScaLAPACK routine library supporting the ILP64 interface	Cluster support	
libmkl_cdft_core.a	Cluster version of FFT functions	Cluster support	
libmkl_blacs_mpich_lp64.a	LP64 version of BLACS routines for MPICH	Cluster support	
libmkl_blacs_mpich_ilp64.a	ILP64 version of BLACS routines for MPICH	Cluster support	

Dynamic Libraries in the `lib` directory

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

File	Contents	Optional Component	
		Name	Installed by Default
libmkl_rt.dylib	Single Dynamic Library		
Interface Layer			
libmkl_intel.dylib	Interface library for the Intel compilers. To be used on IA-32 architecture systems		
libmkl_intel_lp64.dylib	Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support LP64 interface		
libmkl_intel_ilp64.dylib	Interface library for the Intel compilers. To be used on Intel® 64 architecture systems to support ILP64 interface		
Threading Layer			
libmkl_intel_thread.dylib	OpenMP threading library for the Intel compilers		

File	Contents	Optional Component	
		Name	Installed by Default
libmkl_tbb_thread.dylib	Intel TBB threading library for the Intel compilers	Intel TBB threading support	Yes
libmkl_sequential.dylib	Sequential library		
Computational Layer			
libmkl_core.dylib	Contains the dispatcher for dynamic load of the processor-specific kernel library		
libmkl_lapack.dylib	Routines and drivers for LAPACK, DSS, and Intel MKL PARDISO		
libmkl_mc.dylib	64-bit kernel for processors based on the Intel® Core™ microarchitecture		
libmkl_mc3.dylib	64-bit kernel for the Intel® Core™ i7 processors		
libmkl_p4m.dylib	32-bit kernel for the Intel® Core™ microarchitecture		
libmkl_p4m3.dylib	32-bit kernel library for the Intel® Core™ i7 processors		
libmkl_avx.dylib	Kernel library for Intel® Advanced Vector Extensions (Intel® AVX)		
libmkl_avx2.dylib	Kernel library for Intel® Advanced Vector Extensions 2 (Intel® AVX2)		
libmkl_vml_mc.dylib	64-bit Vector Mathematics (VM)/ Vector Statistics (VS)/ Data Fitting (DF) for processors based on the Intel® Core™ microarchitecture		
libmkl_vml_mc2.dylib	64-bit VM/VS/DF for 45nm Hi-k Intel® Core™2 and the Intel Xeon® processor families		
libmkl_vml_mc3.dylib	64-bit VM/VS/DF for the Intel® Core™ i7 processors		

File	Contents	Optional Component	
		Name	Installed by Default
libmkl_vml_p4m.dylib	32-bit VM/VS/DF for processors based on Intel® Core™ microarchitecture		
libmkl_vml_p4m2.dylib	32-bit VM/VS/DF for 45nm Hi-k Intel® Core™2 and Intel Xeon® processor families		
libmkl_vml_p4m3.dylib	32-bit VM/VS/DF for the Intel® Core™ i7 processors		
libmkl_vml_avx.dylib	VM/VS/DF optimized for the Intel® Advanced Vector Extensions (Intel® AVX)		
libmkl_vml_avx2.dylib	VM/VS/DF optimized for Intel® Advanced Vector Extensions 2 (Intel® AVX2)		
libmkl_vml_cmpt.dylib	VM/VS/DF library for conditional numerical reproducibility		
Cluster Libraries			
libmkl_scalapack_lp64.dylib	ScaLAPACK routine library supporting the LP64 interface	Cluster support	
libmkl_scalapack_ilp64.dylib	ScaLAPACK routine library supporting the ILP64 interface	Cluster support	
libmkl_cdft_core.dylib	Cluster version of FFT functions	Cluster support	
libmkl_blacs_mpich_lp64.dylib	LP64 version of BLACS routines for MPICH	Cluster support	
libmkl_blacs_mpich_ilp64.dylib	ILP64 version of BLACS routines for MPICH	Cluster support	
Message Catalogs			
locale/en_US/mkl_msg.cat	Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English		

Index

A

aligning data, example77
architecture support23

B

BLAS
calling routines from C64
Fortran 95 interface to62
OpenMP* threaded routines43

C

C interface to LAPACK, use of64
C, calling LAPACK, BLAS, CBLAS from64
C/C++, Intel(R) MKL complex types65
calling
BLAS functions from C66
CBLAS interface from C66
complex BLAS Level 1 function from C66
complex BLAS Level 1 function from C++66
Fortran-style routines from C64
CBLAS interface, use of64
Cluster FFT
environment variable for92
linking with85
managing performance of92
cluster software, Intel(R) MKL85
cluster software, linking with
commands85
linking examples89
Cluster Sparse Solver, linking with85
code examples, use of21
coding
data alignment77
techniques to improve performance58
compilation, Intel(R) MKL version-dependent78
compiler run-time libraries, linking with38
compiler-dependent function63
complex types in C and C++, Intel(R) MKL65
computation results, consistency69
conditional compilation78
consistent results69
conventions, notational15
custom dynamically linked shared library
building38
composing list of functions39
specifying function names40

D

data alignment, example77
denormal number, performance59
direct call, to Intel(R) Math Kernel Library computational
kernels56
directory structure
high-level23
in-detail
dispatch Intel(R) architectures, configure with an
environment variable93
dispatch, new Intel(R) architectures, enable with an
environment variable93

E

Enter index keyword27
environment variables
for threading control50
setting for specific architecture and programming
interface19
to control dispatching for Intel(R) architectures93
to control threading algorithm for ?gemm53
to enable dispatching of new architectures93
to manage behavior of function domains91
to manage behavior of Intel(R) Math Kernel Library
with91
to manage performance of cluster FFT92
examples, linking
for cluster software89

F

FFT interface
OpenMP* threaded problems43
optimised radices59
FFTW interface support105
Fortran 95 interface libraries36
function call information, enable printing81

H

header files, Intel(R) MKL102
HT technology, configuration tip55

I

ILP64 programming, support for34
improve performance, for matrices of small sizes56
include files, Intel(R) MKL102
information, for function call , enable printing81
installation, checking19
Intel(R) Hyper-Threading Technology, configuration tip55
Intel® Threading Building Blocks, functions threaded
with45
interface
Fortran 95, libraries36
LP64 and ILP64, use of34
interface libraries and modules, Intel(R) MKL61
interface libraries, linking with34

K

kernel, in Intel(R) Math Kernel Library, direct call to56

L

language interfaces support101
language-specific interfaces
interface libraries and modules61
LAPACK
C interface to, use of64
calling routines from C64

- Fortran 95 interface to62
- OpenMP* threaded routines43
- performance of packed routines58
- layers, Intel(R) MKL structure24
- libraries to link with
 - interface34
 - run-time38
 - system libraries38
 - threading36
- link tool, command line29
- link-line syntax32
- linking examples
 - cluster software89
- linking with
 - compiler run-time libraries38
 - interface libraries34
 - system libraries38
 - threading libraries36
- linking, quick start27
- linking, Web-based advisor29
- LINPACK benchmark97

M

- memory functions, redefining60
- memory management59
- memory renaming60
- message-passing interface
 - custom, usage88
 - Intel(R) Math Kernel Library interaction with87
- mixed-language programming63
- module, Fortran 9562
- MPI
 - custom, usage88
 - Intel(R) Math Kernel Library interaction with87

N

- notational conventions15
- number of threads
 - changing at run time47
 - changing with OpenMP* environment variable47
 - Intel(R) MKL choice, particular cases51
 - setting for cluster86
 - techniques to set46
- numerically reproducible results69

O

- OpenMP* threaded functions43
- OpenMP* threaded problems43

P

- parallel performance46

- parallelism, of Intel(R) MKL43
- performance
 - with denormals59
 - with subnormals59
- performance improvement, for matrices of small sizes56
- performance, of Intel(R) MKL, improve on specific processors59

R

- results, consistent, obtaining69
- results, numerically reproducible, obtaining69

S

- ScaLAPACK, linking with85
- SDL28, 33
- Single Dynamic Library28, 33
- structure
 - high-level23
 - in-detail
 - model24
- support, technical9
- supported architectures23
- syntax, link-line32
- system libraries, linking with38

T

- technical support9
- thread safety, of Intel(R) MKL43
- threaded functions, with Intel® Threading Building Blocks45
- threading control, Intel(R) MKL-specific49
- threading libraries, linking with36

U

- unstable output, getting rid of69

V

- Vector Mathematics
 - default mode, setting with environment variable91
 - environment variable to set default mode91
- verbose mode, of Intel(R) MKL81

X

- Xcode*, configuring95