Intel® Math Kernel Library for Linux*

Developer Guide

Intel® MKL2020 - Linux*

Revision: 068

Legal Information

Contents

Appendix A: Appendix A: Intel(R) Math Kernel Library Language Interfaces Support

Appendix B: Support for Third-Party Interfaces

Appendix C: Appendix C: Directory Structure In Detail

Notices and Disclaimers

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

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

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.

Intel, the Intel logo, Intel Atom, Intel Core, Intel Xeon Phi, 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.

© 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.

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

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

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

NOTE
Intel® MKL provides limited input validation to minimize the performance overheads. 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. Examples of the inputs that may result in unexpected behavior:

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

As the Intel® MKL API accepts raw pointers, it is your application's responsibility to validate the buffer sizes before passing them to the library. The library requires subroutine and function parameters to be valid before being passed. While some 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.

Notice revision #20110804

What's New

This Developer Guide documents Intel® Math Kernel Library (Intel® MKL)2020.

The Developer Guide has been updated to fix inaccuracies in the document.

Notational Conventions

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

Linux*	This term refers to information that is valid on all supported Linux* 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`	Indicates filenames, directory names, and pathnames, for example: `./benchmarks/ linpack`

`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,

- 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 forLinux* OS Release Notes*.

Getting Started

1

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

   mkl-32.sh

   mkl.csh

   Use these files to assign Intel® MKL-specific values to several environment variables, as explained in 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

See Also
Setting the Number of Threads Using an OpenMP* Environment Variable

Scripts to Set Environment Variables

When the installation of Intel® MKL forLinux* is complete, set the `INCLUDE, MKLROOT, LD_LIBRARY_PATH,` `MANPATH, LIBRARY_PATH, CPATH,` `NLSPATH,` and `PKG_CONFIG_PATH` environment variables in the command shell using one of the script files in the `bin`subdirectory of the Intel® MKL installation directory. The environment variable `MIC_LD_LIBRARY_PATH` specifies locations of shared objects for Intel® Many Integrated Core Architecture (Intel® MIC Architecture).

Choose the script corresponding to your command shell:

Shell	Script File
C	`mklvars.csh`
Bash and Bourne (sh)	`mklvars.csh`

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

Setting Specified	Required (Yes/No)	Possible Values	Comment
Architecture	Yes, when applicable	`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 intel64`
  sets the environment for Intel® MKL to use the Intel 64 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
Intel® MKL Interface Libraries and Modules
Fortran 95 Interfaces to LAPACK and BLAS
Setting the Number of Threads Using an OpenMP* Environment Variable

Modulefiles to Set Environment Variables

As an alternative, you can use modulefiles to set the environment on Linux*. When the installation of Intel® MKL for Linux* is complete, you need to set some environment variables. To use modulefiles to setup the environment, tell your module utility where they are:

```
$module use /path/to/<mkl_directory>/bin
```

Load modulefile:

Architecture	Command
For Intel® MKL IA-32 architecture	`$module load mkl-32`
For Intel® MKL Intel® 64 architecture	`$module load mkl`

> **NOTE** All existing variables will retain their original values and new values will be prepended to the following variables: *LD_LIBRARY_PATH*, *LIBRARY_PATH*, *CPATH*, *PKG_CONFIG_PATH*, *NLSPATH*, *TBBROOT*. Since *MKLROOT* should contain only one path, it will be set to the new value and any old value of this variable will be lost.

Unloading the module resets the modified environment variables to their original values and new variables that were created will be undefined. *MKLROOT* will be undefined on module unload.

Architecture	Command
For Intel® MKL IA-32 architecture	`$module unload mkl-32`
For Intel® MKL Intel® 64 architecture	`$module unload mkl`

Automating the Process of Setting Environment Variables

To automate setting of the `INCLUDE`, `MKLROOT`, `LD_LIBRARY_PATH`, `MANPATH`, `LIBRARY_PATH`, `CPATH`, and `NLSPATH` environment variables, add `mklvars.*sh` to your shell profile so that each time you login, the script automatically executes and sets the paths to the appropriate Intel® MKL directories. To do this, with a local user account, edit the following files by adding the appropriate script to the path manipulation section right before exporting variables:

Shell	Files	Commands

bash	`~/.bash_profile,`
	`~/.bash_login`
	or `~/.profile`
		`# setting up MKL environment for bash`
		
		`. <absolute_path_to_installed_MKL>/bin`
		`/mklvars.sh [<arch>] [mod] [lp64|ilp64]`

sh	`~/.profile`	`# setting up MKL environment for sh`
		
		`. <absolute_path_to_installed_MKL>/bin`
		`/mklvars.sh [<arch>] [mod] [lp64|ilp64]`

csh	`~/.login`	`# setting up MKL environment for sh`
		
		`. <absolute_path_to_installed_MKL>/bin`
		`/mklvars.csh [<arch>] [mod] [lp64|ilp64]`

In the above commands, the architecture parameter `<arch>` is one of {`ia32|intel64|mic`}.

If you have super user permissions, add the same commands to a general-systemfile in `/etc/profile`(for bash and sh) or in `/etc/csh.login`(for csh).

Caution

Before uninstalling Intel® MKL, remove the above commands from all profile files where the script execution was added. Otherwise you may experience problems logging in.

See Also
Scripts to Set Environment Variables

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 for either C or Fortran, you can select a compiler: Intel®, GNU*, or PGI*.

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
Intel® MKL Include Files

Using Code Examples

The Intel® MKL package includes code examples, located in theexamples 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, theblas subdirectory (extracted from the examples_core archive) contains a makefile to build the BLAS examples and the vmlc subdirectory contains the makefile to build the C examples for Vector Mathematics functions. Source code for the examples is in the next-level sources subdirectory.

See Also
High-level Directory Structure

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

Target platform	Identify the architecture of your target machine: • IA-32 or compatible • Intel® 64 or compatible **Reason:**Because Intel® MKL libraries are located in directories corresponding to your particular architecture (seeArchitecture Support), you should provide proper paths on your link lines (see Linking Examples).To configure your development

environment for the use with Intel® MKL, set your environment variables using the script corresponding to your architecture (see Scripts to Set Environment Variables Setting Environment Variables for details).

Mathematical problem	Identify all Intel® MKL function domains that you require:

- 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, seeImproving 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.

MPI used   Decide what MPI you will use with the Intel® MKL cluster software. You are strongly encouraged to use the latest available version of Intel® MPI.

Reason: To link your application with ScaLAPACK and/or Cluster FFT, the libraries corresponding to your particular MPI should be listed on the link line (see Working with the Intel® Math Kernel Library Cluster Software).

Structure of the Intel® Math Kernel Library

2

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

Architecture Support

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

Architecture	Location
IA-32 or compatible	`<mkl directory>`/lib/ia32_lin
Intel® 64 or compatible	`<mkl directory>`/lib/intel64_lin

See Also
High-level Directory Structure
Notational Conventions
Detailed Structure of the IA-32 Architecture Directory lib/ia32
Detailed Structure of the Intel® 64 Architecture Directory lib/intel64

High-level Directory Structure

Directory	Contents
`<mkl directory>`	Installation directory of the Intel® Math Kernel Library (Intel® MKL)
Subdirectories of `<mkl directory>`	
`bin`	Scripts and modulefiles to set environmental variables in the user shell
`bin/pkgconfig`	Pkg-config files for the most popular configurations of Intel® Math Kernel Library (Intel® MKL)
`benchmarks/linpack`	Shared-memory (SMP) version of the LINPACK benchmark
`benchmarks/mp_linpack`	Message-passing interface (MPI) version of the LINPACK benchmark
`benchmarks/hpcg`	Intel® High Performance Conjugate Gradient Benchmark (Intel® HPCG)

Directory	Contents
examples	Source and data files for Intel® MKL examples. Provided in archives corresponding to Intel® MKL components selected during installation.
include	Include files for the library routines and examples
include/ia32	Fortran 95 .mod files for the IA-32 architecture and Intel® Fortran compiler
include/intel64/lp64	Fortran 95 .mod files for the Intel® 64 architecture, Intel Fortran compiler, and LP64 interface
include/intel64/ ilp64	Fortran 95 .mod files for the Intel® 64 architecture, Intel Fortran compiler, and ILP64 interface
include/fftw	Header files for the FFTW2 and FFTW3 interfaces
interfaces/blas95	Fortran 95 interfaces to BLAS and a makefile to build the library
interfaces/ fftw2x_cdft	MPI FFTW 2.x interfaces to the Intel® MKL Cluster FFT
interfaces/ fftw3x_cdft	MPI FFTW 3.x interfaces to the Intel® MKL Cluster FFT
interfaces/fftw2xc	FFTW 2.x interfaces to the Intel® MKL FFT (C interface)
interfaces/fftw2xf	FFTW 2.x interfaces to the Intel® MKL FFT (Fortran interface)
interfaces/fftw3xc	FFTW 3.x interfaces to the Intel® MKL FFT (C interface)
interfaces/fftw3xf	FFTW 3.x interfaces to the Intel® MKL FFT (Fortran interface)
interfaces/lapack95	Fortran 95 interfaces to LAPACK and a makefile to build the library
lib/ia32_lin	Static libraries and shared objects for the IA-32 architecture
lib/intel64_lin	Static libraries and shared objects for the Intel® 64 architecture
tools	Tools and plug-ins
tools/builder	Tools for creating custom dynamically linkable libraries

See Also
Notational Conventions
Using Code Examples

Layered Model Concept

Intel® 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: • LP64 and ILP64 interfaces. • Compatibility with compilers that return function values differently.
Threading Layer	This layer: • Provides a way to link threaded Intel® MKL with supported compilers. • Enables you to link with a threaded or sequential mode of the library. This layer is compiled for different environments (threaded or sequential) and compilers (from Intel, GNU*, and PGI*).
Computational Layer	This layer accommodates multiple architectures through identification of architecture features and chooses the appropriate binary code at run time.

See Also

Using the ILP64 Interface vs. LP64 Interface
Linking Your Application with the Intel® Math Kernel Library
Linking with Threading Libraries

Linking Your Application with the Intel® Math Kernel 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

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 pkg-config tool to get compilation and link lines	see Using pkg-config metadata files for a summary on how to use Intel® MKL pkg-config metadata files.
Using an interactive interface	see Using the Link-line Advisor to determine libraries and options to specify on your link or compilation line.
Using an internally provided tool	see Using the Command-line Link Tool to determine libraries, options, and environment variables or even compile and build your application.

Using the –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 seeLinking 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
   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.so` on your link line. For example:

```
icc application.c -lmkl_rt
```

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 interface on systems based on the IA-32 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 sectionDynamically 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
Intel® 64 architecture, static linking	`libmkl_intel_ lp64.a`	`libmkl_intel_ thread.a`	`libmkl_core.a`	`libiomp5.so`

	Interface layer	**Threading layer**	**Computational layer**	**RTL**
Intel® 64 architecture, dynamic linking	`libmkl_intel_ lp64.so`	`libmkl_intel_ thread.so`	`libmkl_core. so`	`libiomp5.so`
Intel® Many Integrated Core Architecture (Intel® MIC Architecture), static linking	`libmkl_intel_ lp64.a`	`libmkl_intel_ thread.a`	`libmkl_core.a`	`libiomp5.so`
Intel MIC Architecture, dynamic linking	`libmkl_intel_ lp64.so`	`libmkl_intel_ thread.so`	`libmkl_core.so`	`libiomp5.so`

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

	SDL	**RTL**
Intel® 64 architecture	`libmkl_rt.so`	`libiomp5.so`[††]

[††]Use the Link-line Advisor to check whether you need to explicitly link the `libiomp5.so` 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
Working with the Cluster Software

Using the Link-line Advisor

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

The latest version of the tool is available at http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor. The tool is also available in the documentation directory of the product.

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

See Also
High-level Directory Structure

Using the Command-line Link Tool

Use the command-line Link tool provided by Intel® MKL to simplify building your application with Intel® MKL.

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

The tool `mkl_link_tool` is installed in the *`<mkl directory>`*`/tools` directory, and supports the modes described in the following table.

Intel MKL Command-line Link Tool Modes

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

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

Linking Examples

See Also

Using the Link-line Advisor

Examples for Linking with ScaLAPACK and Cluster FFT

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=$MKLROOT/lib/ia32_lin`,
`MKLINCLUDE=$MKLROOT/include`.

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

- Static linking of `myprog.f` and OpenMP* threadedIntel® MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
-Wl,--end-group -liomp5 -lpthread -lm
```
- Dynamic linking of `myprog.f` and OpenMP* threadedIntel® MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-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
-Wl,--start-group $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a
-Wl,--end-group -lpthread -lm
```
- Dynamic linking of `myprog.f`and sequential version of Intel® MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-lmkl_intel -lmkl_sequential -lmkl_core -lpthread -lm
```
- Dynamic linking of `myprog.f` and OpenMP* threadedor 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 -lmkl_rt
```
- Static linking of `myprog.f`, Fortran 95 LAPACK interface, and OpenMP* threadedIntel® MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/ia32
-lmkl_lapack95
-Wl,--start-group $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
-Wl,--end-group
-liomp5 -lpthread -lm
```
- Static linking of `myprog.f`, Fortran 95 BLAS interface, and OpenMP* threadedIntel® MKL:

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/ia32
-lmkl_blas95
-Wl,--start-group $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a
-Wl,--end-group -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 -lmkl_intel -lmkl_tbb_thread -lmkl_core -ltbb
-lstdc++ -lpthread -lm
```

See Also
Fortran 95 Interfaces to LAPACK and BLAS
Examples for linking a C application using cluster components
Examples for linking a Fortran application using cluster components
Using the Single Dynamic Library
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=$MKLROOT/lib/intel64_lin`,
`MKLINCLUDE=$MKLROOT/include`.

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

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm
```

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-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
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_sequential.a
$MKLPATH/libmkl_core.a -Wl,--end-group -lpthread -lm
```

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

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

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel_ilp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm
```

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
```

```
-lmkl_intel_ilp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm
```
- Dynamic linking of user code `myprog.f` and OpenMP* threadedor sequential Intel® MKL(Call appropriate functions or set environment variables to choose threaded or sequential mode and to set the interface):

  ```
  ifort myprog.f -lmkl_rt
  ```
- Static linking of `myprog.f`, Fortran 95 LAPACK interface, and OpenMP* threadedIntel® MKL supporting the LP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/intel64/lp64
  -lmkl_lapack95_lp64 -Wl,--start-group $MKLPATH/libmkl_intel_lp64.a $MKLPATH/
  libmkl_intel_thread.a
  $MKLPATH/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm
  ```
- Static linking of `myprog.f`, Fortran 95 BLAS interface, and OpenMP* threadedIntel® MKL supporting the LP64 interface:

  ```
  ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/intel64/lp64
  -lmkl_blas95_lp64 -Wl,--start-group $MKLPATH/libmkl_intel_lp64.a $MKLPATH/
  libmkl_intel_thread.a
  $MKLPATH/libmkl_core.a -Wl,--end-group -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-lmkl_intel_lp64 -lmkl_tbb_thread -lmkl_core
  -ltbb -lstdc++ -lpthread -lm
  ```

See Also

Fortran 95 Interfaces to LAPACK and BLAS
Examples for linking a C application using cluster components
Examples for linking a Fortran application using cluster components
Using the Single Dynamic Library
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>intel64|{ilp64|lp64}}]
```

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

```
[<cluster components>]
```

```
-lmkl_{intel|intel_ilp64|intel_lp64|intel_sp2dp|gf|gf_ilp64|gf_lp64}
```

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

```
-lmkl_core
```

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

In the case of static linking,enclose the cluster components, interface, threading, and computational libraries in grouping symbols (for example, `-Wl,--start-group $MKLPATH/libmkl_cdft_core.a $MKLPATH/libmkl_blacs_intelmpi_ilp64.a $MKLPATH/libmkl_intel_ilp64.a $MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group`).

The order of listing libraries on the link line is essential, except for the libraries enclosed in the grouping symbols above.

See Also
Using the Link Line Advisor
Linking Examples
Working with the Cluster Software

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.

The following table lists available interface layers for Intel® 64 architecture along with the values to be used to set each layer.

Specifying the Interface Layer for Intel® 64 Architecture

Interface Layer	Value of `MKL_INTERFACE_LAYER`	Value of the Parameter of `mkl_set_interface_layer`
Intel LP64, default	`LP64`	`MKL_INTERFACE_LP64`
Intel ILP64	`ILP64`	`MKL_INTERFACE_ILP64`
GNU* LP64	`GNU,LP64`	`MKL_INTERFACE_LP64+MKL_INTERFACE_GNU`
GNU ILP64	`GNU,ILP64`	`MKL_INTERFACE_ILP64+MKL_INTERFACE_GNU`

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.

The following table lists available interface layers for IA-32 architecture along with the values to be used to set each layer.

Specifying the Interface Layer for IA-32 Architecture

Interface Layer	Value of `MKL_INTERFACE_LAYER`	Value of the Parameter of `mkl_set_interface_layer`
Intel, default	LP64	MKL_INTERFACE_LP64
GNU	GNU,LP64	MKL_INTERFACE_LP64+MKL_INTERFACE_GNU
	or	or
	GNU	MKL_INTERFACE_GNU

Setting the Threading Layer

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

Specifying the Threading Layer

Threading Layer	Value of `MKL_THREADING_LAYER`	Value of the Parameter of `mkl_set_threading_layer`
Intel threading, default	INTEL	MKL_THREADING_INTEL
Sequential mode of Intel® MKL	SEQUENTIAL	MKL_THREADING_SEQUENTIAL
GNU threading[†]	GNU	MKL_THREADING_GNU
PGI threading[†]	PGI	MKL_THREADING_PGI
Intel TBB threading	TBB	MKL_THREADING_TBB

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

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

See the *Intel® MKL 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.so` or `libmkl_intel_ilp64.so` 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	`ifort –i8 -I<mkl directory>/include ...`
Compiling for LP64	`ifort -I<mkl directory>/include ...`

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

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

Coding for ILP64

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

To migrate to ILP64 or write new code for ILP64, use appropriate types for parameters of the Intel® 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: • 64-bit for ILP64 • 32-bit otherwise	`INTEGER` without specifying `KIND`	`MKL_INT`
Universal integers for ILP64/LP64: • 64-bit integers	`INTEGER*8` or `INTEGER(KIND=8)`	`MKL_INT64`
FFT interface integers for ILP64/LP64	`INTEGER` without specifying `KIND`	`MKL_LONG`

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

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

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

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

  The `viRngUniformBits64` generator function and `vslSkipAheadStream` service function are 64-bit.
- **Summary Statistics:** The *estimate* parameter of the `vslsSSCompute/vsldSSCompute` function is 64-bit.

Refer to the *Intel® MKL 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
Intel® MKL 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 onethreading RTL to your application. Threading RTLs are provided by your compiler. Intel® MKL provides several threading libraries, each dependent on thethreading RTL of a certain compiler, and your choice of the Intel® MKL threading library must be consistent with thethreading RTL that you use in your application.

The OpenMP RTL of the Intel® compiler is the `libiomp5.so` 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.so` 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 anthreading 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 whatthreading 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.so`
				`libiomp5.so`

no	any compiler	parallel	Static linking:
			
			`libmkl_tbb_thread.a`
			
			Dynamic linking:
			
			`libmkl_tbb_thread.so`
				`libtbb.so`

no	any compiler	sequential	Static linking:
			
			`libmkl_sequential.a`
			
			Dynamic linking:
			
			`libmkl_sequential.so`
				none[†]

yes	Intel compiler	parallel	Static linking:
			
			`libmkl_intel_thread.a`
			
			Dynamic linking:
			
			`libmkl_intel_thread.so`
				`libiomp5.so`

yes	GNU compiler	parallel	Static linking: `libmkl_intel_`	Recommended! `libiomp5.so`

Application		Intel® MKL		RTL Required
Uses OpenMP	**Compiled with**	**Execution Mode**	**Threading Layer**	
			`thread.a`	
			Dynamic linking:	
			`libmkl_intel_ thread.so`	
yes	GNU compiler	parallel	Static linking:	GNU OpenMP RTL
			`libmkl_gnu_ thread.a`	
			Dynamic linking:	
			`libmkl_gnu_ thread.so`	
yes	PGI* compiler	parallel	Static linking:	PGI OpenMP RTL
			`libmkl_pgi_ thread.a`	
			Dynamic linking:	
			`libmkl_pgi_ thread.so`	
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.so` libraries depend on `libpthread`.

See Also
Layered Model Concept
Notational Conventions

Linking with Computational Libraries

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

Static Linking	Dynamic Linking
`libmkl_core.a`	`libmkl_core.so`

Computational Libraries for Applications that Use ScaLAPACK or Cluster FFT

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

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

Computational Libraries for IA-32 Architecture

Function domain	Static Linking	Dynamic Linking
ScaLAPACK [†]	`libmkl_scalapack_core.a`	`libmkl_scalapack_core.so`

Function domain	Static Linking	Dynamic Linking
	`libmkl_core.a`	`libmkl_core.so`
Cluster Fourier Transform Functions[†]	`libmkl_cdft_core.a`	`libmkl_cdft_core.so`
	`libmkl_core.a`	`libmkl_core.so`

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

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

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

Function domain	Static Linking	Dynamic Linking
ScaLAPACK, LP64 interface[‡]	`libmkl_scalapack_lp64.a`	`libmkl_scalapack_lp64.so`
	`libmkl_core.a`	`libmkl_core.so`
ScaLAPACK, ILP64 interface[‡]	`libmkl_scalapack_ilp64.a`	`libmkl_scalapack_ilp64.so`
	`libmkl_core.a`	`libmkl_core.so`
Cluster Fourier Transform Functions[‡]	`libmkl_cdft_core.a`	`libmkl_cdft_core.so`
	`libmkl_core.a`	`libmkl_core.so`

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

See Also
Linking with ScaLAPACK and Cluster FFT
Using the Link-line Advisor
Using the ILP64 Interface vs. LP64 Interface

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 `LD_LIBRARY_PATH` environment variable is defined correctly.

See Also
Scripts to Set 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 Shared Objects

Custom shared objectsreduce 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 customshared object builder enables you to create a dynamic library (shared object) 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.

> **NOTE**
> The objects in Intel® MKL static libraries are position-independent code (PIC), which is not typical for static libraries. Therefore, the custom shared object builder can create a shared object from a subset of Intel® MKL functions by picking the respective object files from the static libraries.

Using the Custom Shared Object Builder

To build a custom shared object, 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
`libia32`	The builder uses static Intel® MKL interface, threading, and core libraries to build a customshared object for the IA-32 architecture.
`libintel64`	The builder uses static Intel® MKL interface, threading, and core libraries to build a customshared object for the Intel® 64 architecture.
`soia32`	The builder uses the single dynamic library `libmkl_rt.so` to build a custom shared object for the IA-32 architecture.
`sointel64`	The builder uses the single dynamic library `libmkl_rt.so` to build a custom shared object for the Intel® 64 architecture.
`help`	The command prints Help on the custom shared object builder

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

Parameter [Values]	Description
`interface = {lp64|ilp64}`	Defines whether to use LP64 or ILP64 programming interfacefor the Intel 64architecture.The default value is `lp64`.

Parameter [Values]	Description
`threading = {parallel| sequential}`	Defines whether to use the Intel® MKL in the threaded or sequential mode. The default value is `parallel`.
`cluster = {yes| no}`	(For `libintel64` only) Specifies whether Intel® MKL cluster components (BLACS, ScaLAPACK and/or CDFT) are needed to build the custom shared object. The default value is `no`.
`blacs_mpi = {intelmpi| mpich2|msmpi}`	Specifies the pre-compiled Intel® MKL BLACS library to use. Ignored if `'cluster=no'`. The default value is `intelmpi`.
`blacs_name = <lib name>`	Specifies the name (without extension) of a custom Intel® MKL BLACS library to use. Ignored if `'cluster=no'`. `'blacs_mpi'` is ignored if `'blacs_name'` was explicitly specified. The default value is `mkl_blacs_<blacs_mpi>_<interface>`.
`mpi = <lib name>`	Specifies the name (without extension) of the MPI library used to build the custom DLL. Ignored if `'cluster=no'`. The default value is `impi`.
`export = <file name>`	Specifies the full name of the file that contains the list of entry-point functions to be included in the shared object. The default name is `user_example_list` (no extension).
`name = <so name>`	Specifies the name of the library to be created. By default, the names of the created library is `mkl_custom.so`.
`xerbla = <error handler>`	Specifies the name of the object file `<user_xerbla>.o` that contains the error handler of the user. The makefile adds this error handler to the library for use instead of the default Intel® MKL error handler `xerbla`. If you omit this parameter, the native Intel® MKL `xerbla` is used. See the description of the `xerbla` function in the Intel® MKL Developer Reference to develop your own error handler.
`MKLROOT = <mkl directory>`	Specifies the location of Intel® MKL libraries used to build the custom shared object. By default, the builder uses the Intel® MKL installation directory.

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

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

In the simplest case, the command line is:

```
make ia32
```

and the missing options have default values. This command creates the `mkl_custom.so` library . The command takes the list of functions from the `user_list` file and uses the native Intel® MKL error handler `xerbla`.

Here is an example of a more complex case:

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

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

See Also
Using the Single Dynamic Library

Composing a List of Functions

To compose a list of functions for a minimal custom shared object 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 shared object, 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 shared object.

> **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 Shared Object

To enable use of your custom shared object in a threaded mode, distribute `libiomp5.so` along with the custom shared object.

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. SeeOpenMP* Threaded Functions and Problems and Functions Threaded with Intel® Threading Building Blocks for lists of threaded functions and problems that can be threaded.

Intel® 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 ofOpenMP threads equal to the number of physical cores on the system.

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

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

See Also
Managing Multi-core Performance

OpenMP* Threaded Functions and Problems

The following Intel® MKL function domains are threaded with the OpenMP* technology:

- Direct sparse solver.
- LAPACK.

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

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

- All Level 3 BLAS and all Sparse BLAS routines except Level 2 Sparse Triangular solvers.
- All Vector Mathematics functions (except service functions).
- FFT.

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

LAPACK Routines

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

The following LAPACK routines are threaded with OpenMP*:

- Linear equations, computational routines:

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

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

Threaded BLAS Level1 and Level2 Routines

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

The following routines are threaded with OpenMP* for Intel® Core™2 Duo and Intel® Core™ i7 processors:

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

Threaded FFT Problems

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

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

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

One-dimensional (1D) transforms

1D transforms are threaded in many cases.

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

Architecture	Conditions
Intel® 64	N is a power of 2, $log_2(N) > 9$, the transform is double-precision out-of-place, and input/output strides equal 1.
IA-32	N is a power of 2, $log_2(N) > 13$, and the transform is single-precision.
	N is a power of 2, $log_2(N) > 14$, and the transform is double-precision.
Any	N is composite, $log_2(N) > 16$, and input/output strides equal 1.

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

Multidimensional transforms

All multidimensional transforms on large-volume data are threaded.

Functions Threaded with Intel® Threading Building Blocks

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

The following Intel® MKL function domains are threaded with Intel® Threading Building Blocks (Intel® TBB):

- LAPACK.

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

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

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

LAPACK Routines

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

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

Intel® MKL PARDISO Steps

Intel® MKL PARDISO is threaded with Intel TBB in the reordering and factorization steps. However, routines performing the solving step are still called sequentially when using Intel TBB.

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.

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

- `mkl_sparse_?_mv` using the general compressed sparse row (CSR) and block sparse row (BSR) matrix formats.
- `mkl_sparse_?_mm` using the general CSR sparse matrix format and both row and column major storage formats for the dense matrix.

Avoiding Conflicts in the Execution Environment

Certain situations can cause conflicts in the execution environment that make the use of threads in Intel® 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: `pthreads` on Linux*).	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: `libmkl_sequential.a` or `libmkl_sequential.so` (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 Intel 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: `libmkl_sequential.a` or `libmkl_sequential.so` (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). The Intel® Distribution for LINPACK* Benchmark section discusses another solution for a Hybrid (OpenMP* + MPI) mode.

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

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, use the appropriate command in the command shell in which the program is going to run, for example:

- For the bash shell, enter:
  ```
  export OMP_NUM_THREADS=<number of threads to use>
  ```
- For the csh or tcsh shell, enter:
  ```
  setenv OMP_NUM_THREADS <number of threads to use>
  ```

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

```
}
omp_set_num_threads(2);
for( i=0; i<SIZE; i++)
{
    for( j=0; j<SIZE; j++)
    {
        a[i*SIZE+j]= (double)(i+j);
        b[i*SIZE+j]= (double)(i*j);
        c[i*SIZE+j]= (double)0;
    }
}
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);
printf("row\ta\tc\n");
for ( i=0;i<10;i++)
{
printf("%d:\t%f\t%f\n", i, a[i*SIZE],
c[i*SIZE]);
}
free (a);
free (b);
free (c);
return 0;
}
```

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

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

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

Using Additional Threading Control

Intel® MKL-specific Environment Variables for OpenMP Threading Control

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

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

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

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

Environment Variable	Support Function	Comment	Equivalent OpenMP* Environment Variable
MKL_NUM_THREADS	mkl_set_num_threads mkl_set_num_threads_local	Suggests the number of OpenMP threads to use.	OMP_NUM_THREADS
MKL_DOMAIN_NUM_THREADS	mkl_domain_set_num_threads	Suggests the number of OpenMP threads for a particular function domain.	

Environment Variable	Support Function	Comment	Equivalent OpenMP* Environment Variable
MKL_DYNAMIC	mkl_set_dynamic	Enables Intel® MKL to dynamically change the number of OpenMP threads.	OMP_DYNAMIC

NOTE

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

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

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

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

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

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

MKL_DYNAMIC

The `MKL_DYNAMIC` environment variable enables Intel® MKL to dynamically change the number of threads.

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

When `MKL_DYNAMIC` is `TRUE`, Intel® MKL may use fewer OpenMP threads than the maximum number you specify.

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

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

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

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

- `OMP_NESTED` to `TRUE`

- `OMP_DYNAMIC` and `MKL_DYNAMIC` to `FALSE`
- `MKL_NUM_THREADS` to some reasonable value

With these settings, Intel® MKL uses `MKL_NUM_THREADS` threads when it is called from the OpenMP parallel region in your program.

In general, set `MKL_DYNAMIC` to `FALSE` only under circumstances that Intel® MKL is unable to detect, for example, to use nested parallelism where the library is already called from a parallel section.

MKL_DOMAIN_NUM_THREADS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
mkl_domain_set_num_threads ( 4, MKL_DOMAIN_BLAS );
```

```
mkl_domain_set_num_threads ( 2, MKL_DOMAIN_FFT );
```

MKL_NUM_STRIPES

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

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

Value of **MKL_NUM_STRIPES**	**Partitioning Algorithm**
1 < *nstripes* < (*max_threads_for_mkl/2*)	2D partitioning with the number of partitions equal to *nstripes*: • Horizontal, for column-major ordering. • Vertical, for row-major ordering.
nstripes = 1	1D partitioning algorithm along the opposite direction of the leading dimension.
nstripes ≥ (*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:
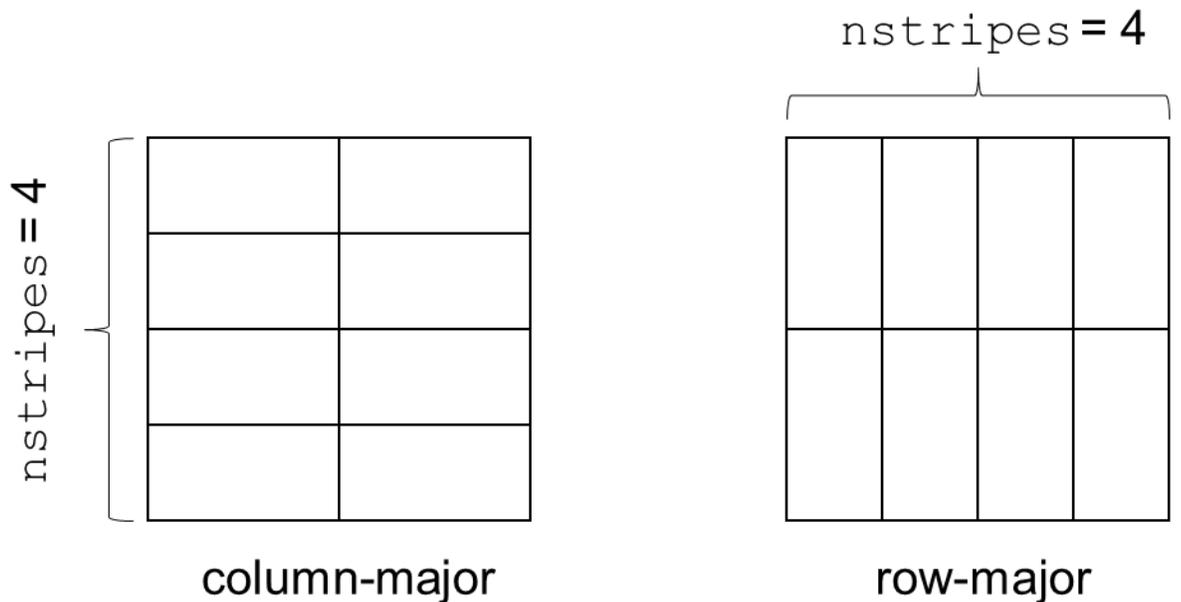
You can use support functions `mkl_set_num_stripes` and `mkl_get_num_stripes` to set and query the number of stripes, respectively.

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 the `export` or `setenv` commands, depending on the shell you use.

For a bash shell, use the `export` commands:

```
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
```

For the csh or tcsh shell, use the `setenv` commands:

```
setenv <VARIABLE NAME><value>.
```

For example:

```
setenv MKL_NUM_THREADS 4
```

```
setenv MKL_DOMAIN_NUM_THREADS "MKL_DOMAIN_ALL=1, MKL_DOMAIN_BLAS=4"
```

```
setenv MKL_DYNAMIC FALSE
```

```
setenv 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 `mkl_rt.so` and select the sequential library using an environment variable or a function call:

    - Set `MKL_THREADING_LAYER=sequential`
    - Call `mkl_set_threading_layer(MKL_THREADING_SEQUENTIAL)`[‡]

Usage model: partition system resources among application threads

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

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

Linking Options:

- Link statically or dynamically with a threading library
- Link with the Single Dynamic Library `mkl_rt.so` and select a threading library using an environment variable or a function call:

    - set `MKL_THREADING_LAYER=intel` or `MKL_THREADING_LAYER=tbb`
    - call `mkl_set_threading_layer(MKL_THREADING_INTEL)` or `mkl_set_threading_layer(MKL_THREADING_TBB)`

Other Options for OpenMP Threading:

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

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

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

    Use if at some point application threads start working with different Intel® MKL function domains.

- Call `mkl_set_num_threads_local()`.

    Use to set the number of OpenMP threads for Intel® MKL called from a particular thread.

NOTE

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

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

---

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

Optimization Notice

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

Notice revision #20110804

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

Using Intel® Hyper-Threading Technology

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

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

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

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

Managing Multi–core Performance

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

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

Consider the following performance issue:

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

To resolve this issue, before calling Intel® MKL, set an affinity mask for each OpenMP thread using theKMP_AFFINITY environment variable or the `sched_setaffinity` system function. The following code example shows how to resolve the issue by setting an affinity mask by operating system means using the Intel compiler. The code calls the function`sched_setaffinity`to bind the threads tothecoreson different sockets. Then the Intel® MKLFFT functionis called:

```
#define _GNU_SOURCE //for using the GNU CPU affinity
// (works with the appropriate kernel and glibc)
// Set affinity mask
#include <sched.h>
#include <stdio.h>
#include <unistd.h>
#include <omp.h>
int main(void) {
    int NCPUs = sysconf(_SC_NPROCESSORS_CONF);
    printf("Using thread affinity on %i NCPUs\n", NCPUs);
#pragma omp parallel default(shared)
    {
        cpu_set_t new_mask;
        cpu_set_t was_mask;
        int tid = omp_get_thread_num();

        CPU_ZERO(&new_mask);

        // 2 packages x 2 cores/pkg x 1 threads/core (4 total cores)
        CPU_SET(tid==0 ? 0 : 2, &new_mask);

        if (sched_getaffinity(0, sizeof(was_mask), &was_mask) == -1) {
            printf("Error: sched_getaffinity(%d, sizeof(was_mask), &was_mask)\n", tid);
        }
        if (sched_setaffinity(0, sizeof(new_mask), &new_mask) == -1) {
            printf("Error: sched_setaffinity(%d, sizeof(new_mask), &new_mask)\n", tid);
        }
        printf("tid=%d new_mask=%08X was_mask=%08X\n", tid,
                        *(unsigned int*)(&new_mask), *(unsigned int*)(&was_mask));
    }
    // Call Intel MKL FFT function
    return 0;
}
```

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

```
icc test_application.c -openmp
```

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

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

```
env OMP_NUM_THREADS=2 ./a.out
```

See the *Linux Programmer's Manual* (in man pages format) for particulars of the `sched_setaffinity`function used in the above example.

Improving Performance for Small Size Problems

The overhead of calling an Intel® MKL function for small problem sizes can be significant when the functionhas a large number of parameters or internally checks parameter errors. To reduce the performance overhead for these small size problems, the Intel® 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	`MKL_DIRECT_CALL`	`-DMKL_DIRECT_CALL`
Sequential	`MKL_DIRECT_CALL_SEQ`	`-DMKL_DIRECT_CALL_SEQ`

- For Fortran applications:

  - Enable preprocessor by using the `-fpp` option for Intel® Fortran Compiler and `–Mpreprocess` option for PGI* compilers.
  - Include the Intel® MKL Fortran include file`mkl_direct_call.fi`.

Intel® MKL skips error checking and intermediate function calls if the problem size is small enough (for example: a 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 -Wl,--start-group $(MKLROOT)/lib/intel64/
libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a
$(MKLROOT)/lib/intel64/libmkl_intel_thread.a -Wl,--end-group -lpthread -lm -openmp -I$(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 -Wl,--start-group $(MKLROOT)/lib/intel64/
libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a
$(MKLROOT)/lib/intel64/libmkl_sequential.a -Wl,--end-group -lpthread -lm -I$(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 sub1()
       stop 1
       end

*      A subroutine that calls DGEMM
       subroutine sub1
*       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 -Wl,--start-group
$(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a $(MKLROOT)/lib/intel64/libmkl_intel_thread.a -Wl,--end-group
-lpthread –lm -openmp -I$(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 -Wl,--start-group
$(MKLROOT)/lib/intel64/libmkl_intel_lp64.a
$(MKLROOT)/lib/intel64/libmkl_core.a $(MKLROOT)/lib/intel64/libmkl_sequential.a -Wl,--end-group
-lpthread –lm -I$(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`, `?getri`, `?potrf`, and `?geqrf`. (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

See Also
Managing Performance of the Cluster Fourier Transform Functions

Coding Techniques

This section discusses coding techniques to improve performance on processors based on supported architectures.

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

Data Alignment and Leading Dimensions

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

For Intel® Xeon Phi™ processor x200 product family, codenamed Knights Landing, align your matrices on 4096-byte boundaries and set the leading dimension to the following integer expression:
$(((n * \textit{element\_size} + 511) / 512) * 512 + 64) /\textit{element\_size}$,
where n is the matrix dimension along the leading dimension.

LAPACK Packed Routines

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

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

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

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

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

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

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

See Also
Managing Performance of the Cluster Fourier Transform Functions

Improving Intel(R) MKL Performance on Specific Processors

Dual–Core Intel® Xeon® Processor 5100 Series

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

Operating on Denormals

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

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

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

Using High-bandwidth Memory with Intel® MKL

To achieve maximum performance, Intel® MKL may use the memkind library (https://github.com/memkind/memkind), which enables controlling memory characteristics and partitioning the heap between different kinds of memory. By default Intel® MKL memory manager tries to allocate memory to Multi-Channel Dynamic Random Access Memory (MCDRAM) using the memkind library on the 2nd generation Intel® Xeon Phi™ product family (for more details of MCDRAM, see https://software.intel.com/en-us/articles/mcdram-high-bandwidth-memory-on-knights-landing-analysis-methods-tools). If allocation of memory to MCDRAM is not possible at the moment, Intel® MKL memory manager falls back to a regular system allocator.

By default the amount of MCDRAM available for Intel® MKL is unlimited. To control the amount of MCDRAM available for Intel® MKL, do either of the following:

- Call

  `mkl_set_memory_limit (MKL_MEM_MCDRAM, <limit_in_mbytes>)`
- Set the environment variable:

  - For the bash shell:

    `MKL_FAST_MEMORY_LIMIT="<limit_in_mbytes>"`
  - For a C shell (csh or tcsh):

    `setenv MKL_FAST_MEMORY_LIMIT "<limit_in_mbytes>"`

The setting of the limit affects all Intel® MKL functions, including user-callable memory functions such as `mkl_malloc`. Therefore, if an application calls `mkl_malloc`, `mkl_calloc`, or `mkl_realloc`, which always tries to allocate memory to MCDRAM, make sure that the limit is sufficient.

If you replace Intel® MKL memory management functions with your own functions (for details, see Redefining Memory Functions), Intel® MKL uses your functions and does not work with the memkind library directly.

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   │
└───┘

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

In addition to the memkind library, 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. You can programmatically redefine values of these pointers to the addresses of your application's memory management functions.

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

How to Redefine Memory Functions

To redefine memory functions, use the following procedure:

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

See Also
Using High-bandwidth Memory with Intel® MKL

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

See Also
Language Interfaces Support, by Function Domain

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.`[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.`[1]	Fortran 95 wrappers for LAPACK (LAPACK95) for IA-32 architecture.

File name	Contains
`libmkl_lapack95_lp64.a`[1]	Fortran 95 wrappers for LAPACK (LAPACK95) supporting LP64 interface.
`libmkl_lapack95_ilp64.a`[1]	Fortran 95 wrappers for LAPACK (LAPACK95) supporting ILP64 interface.
`libfftw2xc_intel.a`[1]	Interfaces for FFTW version 2.x (C interface for Intel compilers) to call Intel® MKL FFT.
`libfftw2xc_gnu.a`	Interfaces for FFTW version 2.x (C interface for GNU compilers) to call Intel® MKL FFT.
`libfftw2xf_intel.a`	Interfaces for FFTW version 2.x (Fortran interface for Intel compilers) to call Intel® MKL FFT.
`libfftw2xf_gnu.a`	Interfaces for FFTW version 2.x (Fortran interface for GNU compiler) to call Intel® MKL FFT.
`libfftw3xc_intel.a`[2]	Interfaces for FFTW version 3.x (C interface for Intel compiler) to call Intel® MKL FFT.
`libfftw3xc_gnu.a`	Interfaces for FFTW version 3.x (C interface for GNU compilers) to call Intel® MKL FFT.
`libfftw3xf_intel.a`[2]	Interfaces for FFTW version 3.x (Fortran interface for Intel compilers) to call Intel® MKL FFT.
`libfftw3xf_gnu.a`	Interfaces for FFTW version 3.x (Fortran interface for GNU compilers) to call Intel® MKL FFT.
`libfftw2x_cdft_SINGLE.a`	Single-precision interfaces for MPI FFTW version 2.x (C interface) to call Intel® MKL cluster FFT.
`libfftw2x_cdft_DOUBLE.a`	Double-precision interfaces for MPI FFTW version 2.x (C interface) to call Intel® MKL cluster FFT.
`libfftw3x_cdft.a`	Interfaces for MPI FFTW version 3.x (C interface) to call Intel® MKL cluster FFT.
`libfftw3x_cdft_ilp64.a`	Interfaces for MPI FFTW version 3.x (C interface) to call Intel® MKL cluster FFT supporting the ILP64 interface.

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

`blas95.mod`[1]	Fortran 95 interface module for BLAS (BLAS95).
`lapack95.mod`[1]	Fortran 95 interface module for LAPACK (LAPACK95).
`f95_precision.mod`[1]	Fortran 95 definition of precision parameters for BLAS95 and LAPACK95.
`mkl_service.mod`[1]	Fortran 95 interface module for Intel® MKL support functions.

[1] Prebuilt for the Intel® Fortran compiler

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

See Also
Fortran 95 Interfaces to LAPACK and BLAS

Fortran 95 Interfaces to LAPACK and BLAS

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

1. Go to the respective directory `<mkl directory>`/interfaces/blas95 or `<mkl directory>`/interfaces/lapack95

2. Type:

- For the IA 32 architecture, `make libia32 INSTALL_DIR` =<user_dir>
- `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 {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, type:

- For the IA-32 architecture, `make cleania32 INSTALL_DIR=<user_dir>`
- `make cleanintel64 [interface=lp64|ilp64] INSTALL_DIR=<user dir>`
- `make clean INSTALL_DIR=<user_dir>`

Caution
Even if you have administrative rights, avoid setting `INSTALL_DIR=../..` or `INSTALL_DIR=<mkl directory>` in a build or clean command above because these settings replace or delete the Intel® 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)
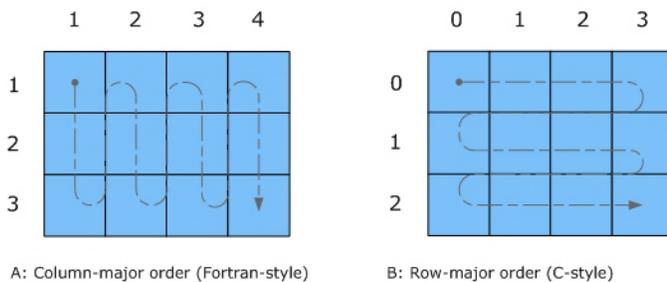
For example, if a two-dimensional matrix A of size `mxn` is stored densely in a one-dimensional array B, you can access a matrix element like this:

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

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

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

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

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

See also the Intel® 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 `mkl.h` header file with the CBLAS interface. The header file specifies enumerated values and prototypes of all the functions. It also determines whether the program is being compiled with a C++ compiler, and if it is, the included file will be correct for use with C++ compilation. Example "Using CBLAS Interface Instead of Calling BLAS Directly from C" illustrates the use of the CBLAS interface.

C Interface to LAPACK

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

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

Using Complex Types in C/C++

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

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

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

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

and

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

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

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

See Also

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

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

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

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

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

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

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

> **NOTE**
> Intel® 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`.

```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:

```cpp
#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:

```c
#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

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

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

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

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

versus

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

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

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

Intel® 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 Extensions2 (Intel® AVX2). The feature-based approach introduces a challenge: if any of the internal floating-point operations are done in a different order or are re-associated, the computed results may differ.

Dispatching optimized code paths based on the capabilities of the processor on which the code is running is central to the optimization approach used by Intel® 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 toReproducibility 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 SSE2instructions:

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

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

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

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

Value	Description
AUTO	CNR mode uses the standard ISA-based dispatching model while ensuring fixed cache sizes, deterministic reductions, and static scheduling
COMPATIBLE	Intel® Streaming SIMD Extensions 2 (Intel® SSE2) without rcpps/rsqrtps instructions
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_2	Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2)
AVX	Intel® Advanced Vector Extensions (Intel® AVX)
AVX2	Intel® Advanced Vector Extensions 2 (Intel® AVX2)
AVX512	Intel AVX-512 on Intel® Xeon® processors
AVX512_E1	Intel® Advanced Vector Extensions 512 (Intel® AVX-512) with support for Vector Neural Network Instructions
AVX512_MIC	Intel® Advanced Vector Extensions 512 (Intel® AVX-512) on Intel® Xeon Phi™ processors
AVX512_MIC_E1	Intel® Advanced Vector Extensions 512 (Intel® AVX-512) with support for Vector Neural Network Instructions on Intel® Xeon Phi™ processors

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

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

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

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

See the *Intel® 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.

Strict CNR Mode

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

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

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

> **NOTE**
>
> - As usual, you should align your data, even in CNR mode, to obtain the best possible performance. While CNR mode also fully supports unaligned input and output data, the use of it might reduce the performance of some 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

```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 7

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

See Also

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

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

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

Example of Data Alignment

Needs for best performance with Intel® 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 Intel 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
`__INTEL_MKL__`	Intel® MKL major version
`__INTEL_MKL_MINOR__`	Intel® MKL minor version
`__INTEL_MKL_UPDATE__`	Intel® MKL update number
`INTEL_MKL_VERSION`	Intel® MKL full version in the following format:
	`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
```

Managing Output

8

Using Intel® MKL Verbose Mode

When building applications that callIntel® MKL functions, it may be useful to determine:

- which computational functions are called,
- what parameters are passed to them, and
- how much time is spent to execute the functions.

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

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

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

- set the environment variable *MKL_VERBOSE* to 1, or
- call the support function `mkl_verbose`(1).

To disable the Intel® MKL Verbose mode,call the `mkl_verbose`(0) function. Both enabling and disabling of the Verbose mode using the function call takes precedence over the environment setting. For a full description of the `mkl_verbose` function, see either the *Intel® MKL Developer Reference for C* or the*Intel® MKL Developer Reference for Fortran.* Both references are available in the Intel® Software Documentation Library.

You can enable Intel® MKL Verbose mode in these domains:

- BLAS
- LAPACK
- ScaLAPACK (selected functionality)
- FFT

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

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

See Also
Intel Software Documentation Library

Version Information Line

In the Intel® 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: • `Lnx` for Linux* OS • `Win` for Windows* OS • `OSX` for macOS*	
The host CPU frequency.		
Intel® MKL interface layer used by the application.	Possible values: • 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`, `gnu_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, Lnx 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 `cblas_` prefix of CBLAS functions is not printed), you can easily recognize the function by the printed name.	
Values of the arguments.	• The values are listed in the order of the formal argument list. The list directly follows the function name, it is parenthesized and comma-separated. • 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 `NULL` references. • Character values are printed without quotes. • For all parameters passed by reference, the values printed are the values *returned by the function*. For example, the printed value of the *info* parameter of a LAPACK function is its value after the function execution.	

Information	Description	Related Links
	• For verbose-enabled functions in the ScaLAPACK domain, in addition to the standard input parameters, information about blocking factors, MPI rank, and process grid is also printed.	
Time taken by the function.	• The time is printed in convenient units (seconds, milliseconds, and so on), which are explicitly indicated. • The time may fluctuate from run to run. • The time printed may occasionally be larger than the time actually taken by the function call, especially for small problem sizes and multi-socket machines.To reduce this effect, bind threads that call Intel® MKL to CPU cores by setting an affinity mask.	Managing Multi-core Performance for options to set an affinity mask.
Value of the `MKL_CBWR` environment variable.	The value printed is prefixed with `CNR`:	Getting Started with Conditional Numerical Reproducibility
Value of the `MKL_DYNAMIC` environment variable.	The value printed is prefixed with `Dyn`:	`MKL_DYNAMIC`
Status of the Intel® MKLmemory manager.	The value printed is prefixed with `FastMM`:	Avoiding Memory Leaks in Intel® MKLfor a description of the Intel® MKLmemory manager
OpenMP* thread number of the calling thread.	The value printed is prefixed with `TID`:	
Values of Intel® MKL environment variables defining the general and domain-specific numbers of threads, separated by a comma.	The first value printed is prefixed with `NThr`:	Intel® MKL-specific Environment Variables for Threading Control

The following is an example of a call description line:

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

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

<div style="float:right">9</div>

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

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

ScaLAPACK, Cluster FFT, and Cluster Sparse Solver are only provided for the Intel® 64 and Intel® Many Integrated Core architectures.

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

Optimization Notice

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

Notice revision #20110804

See Also
Intel® Math Kernel Library Structure
Managing Performance of the Cluster Fourier Transform Functions
Intel® Distribution for LINPACK* Benchmark

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> [-Wl,--start-group] [<MKL cluster library>]       \
<BLACS> <MKL core libraries> [-Wl,--end-group]
```

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

<MKL cluster library>	One of libraries for ScaLAPACK or Cluster FFT and appropriate architecture and programming interface (LP64 or ILP64). Available libraries are listed in Appendix C: Directory Structure in Detail. For example, for the LP64 interface, it is – `lmkl_scalapack_lp64` or `–lmkl_cdft_core`. Cluster Sparse Solver does not require an additional computation library.
<BLACS>	The BLACS library corresponding to your architecture, programming interface (LP64 or ILP64), and MPI used. Available BLACS libraries are listed in Appendix C: Directory Structure in Detail. Specifically, choose one of – `lmkl_blacs_intelmpi_lp64` or – `lmkl_blacs_intelmpi_ilp64`.
<MKL core libraries>	Processor optimized kernels, threading library, and system library for threading support, linked as described in Listing Libraries on a Link Line.
<MPI linker script>	A linker script that corresponds to the MPI version.

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

```
-L$MKLPATH -I$MKLINCLUDE -Wl,--start-group $MKLPATH/libmkl_scalapack_lp64.a $MKLPATH/
libmkl_blacs_intelmpi_lp64.a $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_sequential.a
$MKLPATH/libmkl_core.a -static_mpi -Wl,--end-group -lpthread -lm
```

> **NOTE**
> Grouping symbols `-Wl,--start-group` and `-Wl,--end-group` are required for static linking.

> **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(R) 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 LD_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.

Setting Environment Variables on a Cluster

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

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

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

See these MPICH2 examples on how to set the value of OMP_NUM_THREADS explicitly:

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

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

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

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

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

Interaction with the Message-passing Interface

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

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

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

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

- 0 or undefined.

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

  This value determines Intel® MKL conclusion of the thread support level:

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

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

  Important
  Instead of relying on the discussed implicit settings, explicitly set the number of threads for Intel® MKL.

Optimization Notice

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

Notice revision #20110804

See Also
Managing Multi-core Performance

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

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

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

For example: the command

```
make libintel64
```

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

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

```
make help
```

Using a Custom BLACS Library

Use custom BLACS libraries exactly the same way as you use the prebuilt BLACS libraries, but pass the custom library to the linker. For example, instead of passing the `libmkl_blacs_intelmpi_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 MPICH2 or higher and OpenMPI, and the number of MPI processes is set by –`np`. In the case of MPICH2 or higher and Intel MPI, 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

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

- Main module is in C.
- You are using the Intel® C++ Compiler.
- You are using MPICH2.
- Intel® MKL functions use LP64 interfaces.
- The `PATH` environment variable contains a directory with the MPI linker scripts.
- `$MKLPATH` is a user-defined variable containing `<mkl_directory>`/lib/intel64_lin.

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

```
mpicc <user files to link>                       \
    -L$MKLPATH                                   \
    -lmkl_scalapack_lp64                         \
    -lmkl_blacs_intelmpi_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>               \
    -Wl,--start-group                    \
    $MKLPATH/libmkl_cdft_core.a          \
    $MKLPATH/libmkl_blacs_intelmpi_lp64.a  \
    $MKLPATH/libmkl_intel_lp64.a         \
    $MKLPATH/libmkl_intel_thread.a       \
    $MKLPATH/libmkl_core.a               \
    -Wl,--end-group                      \
    -liomp5 -lpthread
```

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

```
mpicc <user files to link>                         \
    -L$MKLPATH                                     \
    -lmkl_blacs_intelmpi_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 Intel MPI library.
- Intel® MKL functions use LP64 interfaces.
- The `PATH` environment variable contains a directory with the MPI linker scripts.
- `$MKLPATH` is a user-defined variable containing `<mkl_directory>`/lib/intel64_lin.

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

```
mpiifort <user files to link>                       \
    -L$MKLPATH                                       \
    -lmkl_scalapack_lp64                             \
    -lmkl_blacs_intelmpi_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>                       \
    -Wl,--start-group                                \
    $MKLPATH/libmkl_cdft_core.a                      \
    $MKLPATH/libmkl_blacs_intelmpi_lp64.a            \
    $MKLPATH/libmkl_intel_lp64.a                     \
    $MKLPATH/libmkl_intel_thread.a                   \
    $MKLPATH/libmkl_core.a                           \
    -Wl,--end-group                                  \
    -liomp5 -lpthread
```

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>                       \
    -Wl,--start-group                                \
    $MKLPATH/libmkl_blacs_intelmpi_lp64.a            \
    $MKLPATH/libmkl_intel_lp64.a                     \
    $MKLPATH/libmkl_intel_thread.a                   \
    $MKLPATH/libmkl_core.a                           \
    -Wl,--end-group                                  \
    -liomp5 -lpthread
```

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

See Also

Intel® MKL-specific Environment Variables for Threading Control

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

Managing Behavior of Function Domains with Environment Variables

Setting the Default Mode of Vector Math with an Environment Variable

Intel® 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 *mode*	Description
Accuracy Control	
`VML_HA`	high accuracy versions of VM functions

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

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

- For the bash shell:

  `export MKL_VML_MODE=VML_LA,VML_ERRMODE_ERRNO,VML_ERRMODE_STDERR`
- For a C shell (csh or tcsh):

  `setenv MKL_VML_MODE VML_LA,VML_ERRMODE_ERRNO,VML_ERRMODE_STDERR`

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

Managing Performance of the Cluster Fourier Transform Functions

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

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

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

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

Option	Possible Values	Description
`alltoallv`	0 (default)	Configures CFFT to use the standard `MPI_Alltoallv` function to perform global transpositions.

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

> **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 assuming the bash shell:

```
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

Managing Invalid Input Checking in LAPACKE Functions

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

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

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

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

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

Instruction Set Specific Dispatching on Intel® Architectures

Intel® MKL automatically queries and then dispatches the code path supported on your Intel® processor to the optimal instruction set architecture (ISA) by default. The `MKL_ENABLE_INSTRUCTIONS` environment variable or the `mkl_enable_instructions` support function enables you to dispatch to an ISA-specific code path of your choice. For example, you can run the Intel® Advanced Vector Extensions (Intel® AVX) code path on an Intel processor based on Intel® Advanced Vector Extensions 2 (Intel® AVX2), or you can run the Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) code path on an Intel AVX-enabled Intel processor. This feature is not available on non-Intel processors.

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

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

Value of `MKL_ENABLE_INSTRUCTIONS`	ISA	Dispatched by Default
`AVX512`	Intel® Advanced Vector Extensions (Intel® AVX-512) for systems based on Intel® Xeon® processors	Yes
`AVX512_E1`	Intel® Advanced Vector Extensions (Intel® AVX-512) with support for Vector Neural Network Instructions.	Yes
`AVX512_E2`	ICX: Intel® Advanced Vector Extensions (Intel® AVX-512) enabled processors.	Yes
`AVX512_MIC`	Intel® Advanced Vector Extensions (Intel® AVX-512) for systems based on Intel® Xeon Phi™ processors	Yes

Value of `MKL_ENABLE_INSTRUCTIONS`	ISA	Dispatched by Default
`AVX512_MIC_E1`	Intel® Advanced Vector Extensions 512 (Intel® AVX-512) for Intel® Many Integrated Core Architecture (Intel® MIC Architecture) with support for AVX512_4FMAPS and AVX512_4VNNIW instruction groups enabled 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 turn on automatic CPU-based dispatching of Intel AVX-512 with support of AVX512_4FMAPS and AVX512_4VNNI instruction groups on systems based on Intel Xeon Phi processors, do one of the following:
  - Call

    `mkl_enable_instructions(MKL_ENABLE_AVX512_MIC_E1)`
  - Set the environment variable:
    - For the bash shell:

      `export MKL_ENABLE_INSTRUCTIONS=AVX512_MIC_E1`
    - For a C shell (csh or tcsh):

      `setenv MKL_ENABLE_INSTRUCTIONS AVX512_MIC_E1`
- 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

---

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

Configuring Your Integrated Development Environment to Link with Intel® Math Kernel Library

11

Configuring the Eclipse* IDE CDT to Link with Intel® MKL

This section explains how to configure the Eclipse* Integrated Development Environment (IDE) C/C++ Development Tools (CDT) to link with Intel® Math Kernel Library (Intel® MKL).

Tip
After configuring your CDT, you can benefit from the Eclipse-provided *code assist* feature. See Code/Context Assist description in the CDT Help for details.

To configure your Eclipse IDE CDT 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 CDT and on the tool-chain/ compiler integration. Refer to the CDT Help for more details.

To configure your Eclipse IDE CDT, do the following:

1. Open **Project Properties** for your project.
2. Add the Intel® MKL include path, that is, `<mkl directory>/include`, to the project's include paths.
3. Add the Intel® MKL library path for the target architecture to the project's library paths. For example, for the Intel® 64 architecture, add `<mkl directory>/lib/intel64_lin`.
4. Specify the names of the Intel® MKL libraries to link with your application. For example, you may need the following libraries: `mkl_intel_lp64`, `mkl_intel_thread`, `mkl_core`, and `iomp5`.

NOTE
Because compilers typically require library names rather than file names, omit the "`lib`" prefix and "`a`" or "`so`" extension.

See Also
Intel® MKL Libraries to Link with
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 Linux*

Intel® Optimized LINPACK Benchmark for Linux* 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:

- Intel® Distribution for LINPACK* Benchmark, which is a distributed memory version of the same benchmark.
- LINPACK, the library, which has been expanded upon by the LAPACK library.

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

Additional information on this software, as well as on other Intel® software performance products, is available at http://www.intel.com/software/products/.

Acknowledgement

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

Contents of the Intel® Optimized LINPACK Benchmark

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

File in `./benchmarks/linpack/`	**Description**
`xlinpack_xeon32`	The 32-bit program executable for a system based on Intel® Xeon® processor or Intel® Xeon® processor MP with or without Intel® Streaming SIMD Extensions 3 (SSE3).

File in `./benchmarks/linpack/`	**Description**
`xlinpack_xeon64`	The 64-bit program executable for a system with Intel Xeon processor using Intel® 64 architecture.
`runme_xeon32`	A sample shell script for executing a pre-determined problem set for `xlinpack_xeon32`.
`lininput_xeon32`	Input file for a pre-determined problem for the `runme_xeon32` script.
`lininput_xeon64`	Input file for a pre-determined problem for the `runme_xeon64` script.
`help.lpk`	Simple help file.
`xhelp.lpk`	Extended help file.

These files are not available immediately after installation and appear as a result of execution of an appropriate `runme` script.

`lin_xeon32.txt`	Result of the `runme_xeon32` script execution.
`lin_xeon64.txt`	Result of the `runme_xeon64` 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:

```
./runme_xeon32
```

```
./runme_xeon64
```

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

```
./xlinpack_xeon32-e
```

```
./xlinpack_xeon64 -e
```

The pre-defined data input files `lininput_xeon32`, `lininput_xeon64`, are examples. Different systems have different numbers of processors or amounts of memory and therefore require new input files. The extended help can give insight into proper ways to change the sample input files.

Each input file requires the following minimum amount of memory:

`lininput_xeon32`	2 GB
`lininput_xeon64`	16 GB

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

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

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

Known Limitations of the Intel® Optimized LINPACK Benchmark

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

* Intel Optimized LINPACK Benchmark supports only OpenMP threading
* Intel Optimized LINPACK Benchmark is threaded to effectively use multiple processors. So, in multi-processor systems, best performance will be obtained with the Intel® Hyper-Threading Technology turned off, which ensures that the operating system assigns threads to physical processors only.
* If an incomplete data input file is given, the binaries may either hang or fault. See the sample data input files and/or the extended help for insight into creating a correct data input file.

Intel® Distribution for LINPACK* Benchmark

Overview of the Intel Distribution for LINPACK Benchmark

The Intel® Distribution for LINPACK* Benchmark is based on modifications and additions to High-Performance LINPACK (HPL) (http://www.netlib.org/benchmark/hpl/) from Innovative Computing Laboratories (ICL) at the University of Tennessee, Knoxville. The Intel Distribution for LINPACK Benchmark can be used for TOP500 runs (see http://www.top500.org) and for benchmarking your cluster. To use the benchmark you need to be familiar with HPL usage. The Intel Distribution for LINPACK Benchmark provides some enhancements designed to make the HPL usage more convenient and to use Intel® Message-Passing Interface (MPI) settings to improve performance.

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

Intel provides optimized versions of the LINPACK benchmarks to help you obtain high LINPACK benchmark results on your systems based on genuine Intel processors more easily than with the standard HPL benchmark. The prebuilt binaries require Intel® MPI library be installed on the cluster. The run-time version of Intel MPI library is free and can be downloaded from http://www.intel.com/software/products/ .

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

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

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

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

Contents of the Intel Distribution for LINPACK Benchmark

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

File in `<mkl directory>`/benchmarks/ mp_linpack/	**Contents**
`COPYRIGHT`	Original Netlib HPL copyright document.
`readme.txt`	Information about the files provided.
Prebuilt executables for performance testing	
`xhpl_intel64_dynamic`	Prebuilt binary for the Intel® 64 architecture dynamically linked against Intel MPI library[‡].
`xhpl_intel64_static`	Prebuilt binary for the Intel® 64 architecture statically linked against Intel MPI library.
Run scripts and an input file example	
`runme_intel64_dynamic`	Sample run script for the Intel® 64 architecture and binary dynamically linked against Intel MPI library.
`runme_intel64_static`	Sample run script for the Intel® 64 architecture and binary statically linked against Intel MPI library.
`runme_intel64_prv`	Script that sets HPL environment variables. It is called by `runme_intel64_static` and `runme_intel64_dynamic`.
`HPL.dat`	Example of an HPL configuration file.
Prebuilt libraries and utilities for building with a customized MPI implementation	
`libhpl_intel64.a`	Library file required to build Intel Distribution for LINPACK Benchmark for the Intel® 64 architecture with a customized MPI implementation.
`HPL_main.c`	Source code required to build Intel Distribution for LINPACK Benchmark for the Intel® 64 architecture with a customized MPI implementation.
`build.sh`	Build script for creating Intel Distribution for LINPACK Benchmark for the Intel® 64 architecture with a customized MPI implementation.

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

See Also
High-level Directory Structure

Building the Intel Distribution for LINPACK Benchmark for a Customized MPI Implementation

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

1. Specify the location of Intel® MKL to be used (`MKLROOT`)
2. Set up your MPI environment
3. Run the script `build.sh`

See Also
Contents of the Intel Distribution for LINPACK Benchmark

Building the Netlib HPL from Source Code

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

1. Download and extract the source code.
2. Copy the makefile:

```
$> cp setup/Make.Linux_Intel64 .
```
3. Edit `Make.Linux_Intel64` as appropriate
4. Build the HPL binary:

```
$> make arch=Linux_Intel64
```
5. Check that the built binary is available in the `bin/Linux_Intel64` directory.

> **NOTE**
> The Intel Distribution for LINPACK Benchmark may contain additional optimizations compared to the reference Netlib HPL implementation.

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
Contents of the Intel Distribution for LINPACK Benchmark

Configuring Parameters

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

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

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

  Choose $P \leq Q$.

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

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

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

- *N* - the problem size:

  - For homogeneous runs, choose *N* divisible by *NB*LCM(P,Q)*, where *LCM* is the least common multiple of the two numbers.
  - For heterogeneous runs, see Heterogeneous Support in the Intel Distribution for LINPACK Benchmark for how to choose *N*.

NOTE
Increasing *N* usually increases performance, but the size of *N* is bounded by memory. In general, you can compute the memory required to store the matrix (which does not count internal buffers) as $8*N*N/(P*Q)$ bytes, where *N* is the problem size and *P* and *Q* are the process grids in `HPL.dat`. A general rule of thumb is to choose a problem size that fills 80% of memory.

Ease-of-use Command-line Parameters

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

Placeholders in this command line illustrate these parameters:

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

You can also use command-line parameters with the sample runme scripts. For example:

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

For more command-line parameters, see Heterogeneous Support in the Intel Distribution for LINPACK Benchmark.

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

```
./runme_intel64_dynamic -n 10000 -p 1 -q 3
```

By using the `m` parameter you can scale by the memory size instead of the problem size. The `m` parameter only refers to the size of the matrix storage. Therefore, to use matrices that fit in 50000 Mbytes with *NB*=256 on 16 nodes, adjust the script to set the total number of MPI processes to 16 and execute this command:

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

Running the Intel Distribution for LINPACK Benchmark

To run the Intel Distribution for LINPACK Benchmark on multiple nodes or on one node with multiple MPI processes, you need to use MPI and either modify `HPL.dat` or use Ease-of-use Command-line Parameters. The following example describes how to run the dynamically-linked prebuilt Intel Distribution for LINPACK Benchmark binary using the script provided. To run other binaries, adjust the steps accordingly; specifically, change line 58 of `runme_intel64_dynamic` to point to the appropriate binary.

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

`<parent directory>/bin/compilervars.shintel64`

`<mpi directory>/bin64/mpivars.sh`

2. In `HPL.dat`, set the problem size *N* to 10000. Because this setting is for a test run, the problem size should be small.

3. For better performance, enable non-uniform memory access (NUMA) on your system and configure to run an MPI process for each NUMA socket as explained below.

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

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

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

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

4. Execute `runme_intel64_dynamic` script:

`./runme_intel64_dynamic`

5. Rerun the test increasing the size of the problem until the matrix size uses about 80% of the available memory. To do this, either modify `Ns` in line 6 of `HPL.dat` or use the `-n` command-line parameter:

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

See Also
Notational Conventions
Building the Intel Distribution for LINPACK Benchmark for a Customized MPI Implementation
Building the Netlib HPL from Source Code
Using High-bandwidth Memory with Intel® MKL

Heterogeneous Support in the Intel Distribution for LINPACK Benchmark

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

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

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

Let *P* be the number of rows and *Q* the number of columns in your processor grid (*PxQ*). The work must be *homogeneous* within each processor column because vertical operations, such as pivoting or panel factorization, are synchronizing operations. When there are two different types of nodes, use MPI to process all the faster nodes first and make sure the "`PMAP process mapping`" (line 9) of `HPL.dat` is set to 1 for `Column-major` mapping. Because all the nodes must be the same within a process column, the number of faster nodes must always be a multiple of *P*, and you can specify the faster nodes by setting the number of process columns *C* for the faster nodes with the `c` command-line parameter. The `-f 1.0 -c 0` setting corresponds to the default homogeneous behavior.

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

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

Memory Utilization is usually around 0.8 for homogeneous Intel Xeon processor systems. On a heterogeneous system, you may apply a different formula for *N* for each set of nodes that are the same and then choose the minimum *N* over all sets. Suppose you have a cluster with only one heterogeneous factor *F* and the number of processor columns (out of the total *Q*) in the group with that heterogeneous factor equal to *C*. That group contains *P*C* nodes. First compute the sum of the parts: *S =F*P*C + P*(Q-C)*. Note that on a homogeneous system *S=P*Q,F=1*, and *C=Q*. Take *N* as

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

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

Example

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

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

If you redistribute the matrix and run the heterogeneous Intel Distribution for LINPACK Benchmark, you can take advantage of the faster nodes. But because some of the nodes will contain 2.7 times as much data as the other nodes, you must shrink the problem size (unless the faster nodes also happen to have 2.7 times as much memory). Instead of 0.8*64GB*100 total memory size, we have only 0.8*64GB*20 + 0.8*64GB/2.7*80 total memory size, which is less than half the original space. So the problem size in this case would be 526000. Because *P*=10 and there are 20 faster nodes, two processor columns are faster. If you arrange MPI to send these nodes first to the application, the command line looks like:

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

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

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

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

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

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

If instead there are three different types of nodes in a cluster and you need at least two heterogeneous factors, change the number in the first row above from 1 to 2 and follow that line with two lines specifying the start column, stopping column, and heterogeneous factor.

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

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

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

Environment Variables

The table below lists Intel® MKL environment variables to control runs of the Intel Distribution for LINPACK Benchmark.

Environment Variable	Description	Value
HPL_LARGEPAGE	Defines the memory mapping to be used for the Intel Xeon processor.	0 or 1: • 0 - normal memory mapping, default. • 1 - memory mapping with large pages (2 MB per page mapping). It may increase performance.
HPL_LOG	Controls the level of detail for the HPL output.	An integer ranging from 0 to 2: • 0 - no log is displayed. • 1 - only one root node displays a log, exactly the same as the ASYOUGO option provides.

Environment Variable	Description	Value
		• 2 - the most detailed log is displayed. All *P* root nodes in the processor column that owns the current column block display a log.
`HPL_HOST_CORE`, `HPL_HOST_NODE`	Specifies cores or Non-Uniform Memory Access (NUMA) nodes to be used. `HPL_HOST_NODE` requires NUMA mode to be enabled. You can check whether it is enabled by the `numactl --hardware` command. The default behavior is auto-detection of the core or NUMA node.	A list of integers ranging from 0 to the largest number of a core or NUMA node in the cluster and separated as explained in example 3.
`HPL_SWAPWIDTH`	Specifies width for each swap operation.	16 or 24. The default is 24.

You can set Intel Distribution for LINPACK Benchmark environment variables using the `PMI_RANK` and `PMI_SIZE` environment variables of the Intel MPI library, and you can create a shell script to automate the process.

Examples of Environment Settings

#	Settings	Behavior of the Intel Distribution for LINPACK Benchmark
1	Nothing specified	All Intel Xeon processors in the cluster are used.

2	`HPL_MIC_DEVICE=0,2`
	
	`HPL_HOST_CORE=1-3,8-10`
		Intel Xeon processor cores 1,2,3,8,9, and 10 are used.

3	`HPL_HOST_NODE=1`	Only Intel Xeon processor cores on NUMA node 1 are used.

Improving Performance of Your Cluster

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

1. Reboot all nodes.
2. Ensure all nodes are in identical conditions and no zombie processes are left running from prior HPL runs. To do this, run single-node Stream and Intel Distribution for LINPACK Benchmark on every node. Ensure results are within 10% of each other (problem size must be large enough depending on memory size and CPU speed). Investigate nodes with low performance for hardware/software problems.
3. Check that your cluster interconnects are working. Run a test over the complete cluster using an MPI test for bandwidth and latency, such as one found in the Intel® MPI Benchmarks package.
4. Run an Intel Distribution for LINPACK Benchmark on pairs of two or four nodes and ensure results are within 10% of each other. The problem size must be large enough depending on the memory size and CPU speed.
5. Run a small problem size over the complete cluster to ensure correctness.
6. Increase the problem size and run the real test load.
7. In case of problems go back to step 2.

Before making a heterogeneous run, always run its homogeneous equivalent first.

Intel® Optimized High Performance Conjugate Gradient Benchmark

Overview of the Intel Optimized HPCG

The Intel® Optimized High Performance Conjugate Gradient Benchmark (Intel® Optimized HPCG) provides an implementation of the HPCG benchmark (http://hpcg-benchmark.org) optimized for Intel® Xeon® processors and Intel® Xeon Phi™ processors with Intel® Advanced Vector Extensions (Intel® AVX), Intel® Advanced Vector Extensions 2 (Intel® AVX2), Intel® Advanced Vector Extensions 512 (Intel® AVX-512) support. The HPCG Benchmark is intended to complement the High Performance LINPACK benchmark used in the TOP500 (http://www.top500.org) system ranking by providing a metric that better aligns with a broader set of important cluster applications.

The HPCG benchmark implementation is based on a 3-dimensional (3D) regular 27-point discretization of an elliptic partial differential equation. The implementation calls a 3D domain to fill a 3D virtual process grid for all the available MPI ranks. HPCG uses the preconditioned conjugate gradient method (CG) to solve the intermediate systems of equations and incorporates a local and symmetric Gauss-Seidel preconditioning step that requires a triangular forward solve and a backward solve. A synthetic multi-grid V-cycle is used on each preconditioning step to make the benchmark better fit real-world applications. HPCG implements matrix multiplication locally, with an initial halo exchange between neighboring processes. The benchmark exhibits irregular accesses to memory and fine-grain recursive computations that dominate many scientific workloads (for details, see http://www.sandia.gov/~maherou/docs/HPCG-Benchmark.pdf).

The Intel® Optimized HPCG contains source code of the HPCG v3.0 reference implementation with necessary modifications to include:

- Intel® architecture optimizations
- Prebuilt benchmark executables that link to Intel® Math Kernel Library (Intel® MKL)

  - Inspector-executor Sparse BLAS kernels for sparse matrix-vector multiplication (SpMV)

  - Sparse triangular solve (TRSV)

  - Symmetric Gauss-Seidel smoother (SYMGS)

that are optimized for Intel AVX, Intel AVX2, and Intel AVX-512 instruction sets. For the Intel AVX-512 instruction set, there are separate versions that target Intel® Xeon® Scalable processors and Intel® Xeon® Phi processors. Use this package to evaluate the performance of distributed-memory systems based on any generation of the Intel® Xeon® processor E3, Intel® Xeon® processor E5, Intel® Xeon® processor E7, Intel® Xeon® Scalable processor family, and Intel Xeon Phi™ processor families.

The Intel® Math Kernel Library Inspector-executor Sparse BLAS kernels SpMV, TRSV, and SYMGS are implemented using an inspector-executor model. The inspection step chooses the best algorithm for the input matrix and converts the matrix to a special internal representation to achieve high performance at the execution step.

Versions of the Intel Optimized HPCG

The Intel Optimized HPCG package includes prebuilt HPCG benchmark for Intel MPI 5.1 or higher. All the files of the benchmark are located in the `./benchmarks/hpcg`subdirectory of the Intel® MKL directory. These versions of the benchmark are available:

File in `./benchmarks/ hpcg/bin`	**Description**
`xhpcg_avx`	The Intel AVX optimized version of the benchmark, optimized for systems based on the first and the second generations of Intel Xeon processor E3 family, Intel Xeon processor E5 family, or Intel Xeon processor E7 family.
`xhpcg_avx2`	The Intel AVX2 optimized version of the benchmark, optimized for systems based on the third and later generations of the Intel Xeon processor E3 family, Intel Xeon processor E5 family, Intel Xeon processor E7 family, and future Intel processors with Intel AVX2 support. Running the Intel AVX optimized version of the benchmark on an Intel AVX2 enabled system produces non-optimal performance. The Intel AVX2 optimized version of the benchmark does not run on systems that do not support Intel AVX2.
`xhpcg_knl`	The Intel Xeon Phi processor (formerly Knights Landing) optimized version of the benchmark is designed for systems based on Intel Xeon Phi processors with Intel AVX-512 support. Running the Intel AVX or AVX2 optimized versions of the benchmark on an Intel AVX-512 enabled system produces non-optimal performance. The Intel Xeon Phi processor optimized version of the benchmark does not run on systems that do not support Intel AVX-512.
`xhpcg_skx`	The Intel Xeon Scalable processor (formerly Skylake) optimized version of the benchmark is designed for systems based on Intel Xeon Scalable processors and future Intel processors with Intel AVX-512 support. Running the Intel AVX or AVX2 optimized versions of the benchmark on an Intel AVX-512 enabled system produces non-optimal performance. The Intel Xeon Scalable processor optimized version of the benchmark does not run on systems that do not support Intel AVX-512.

The Intel Optimized HPCG package also includes the source code necessary to build these versions of the benchmark for other MPI implementations, such as SGI MPT*, MPICH2, or Open MPI: Intel AVX optimized version and Intel AVX2 optimized version, and Intel AVX-512 optimized version. Build instructions are available in the `QUICKSTART` file included with the package.

See Also
High-level Directory Structure

Getting Started with Intel Optimized HPCG

To start working with the benchmark,

1. On a cluster file system, unpack the Intel Optimized HPCG package to a directory accessible by all nodes. Read and accept the license as indicated in the `readme.txt` file included in the package.
2. Change the directory to `hpcg/bin`.
3. Determine the prebuilt version of the benchmark that is best for your system or follow `QUICKSTART` instructions to build a version of the benchmark for your MPI implementation.
4. Ensure that Intel® MKL, Intel C/C++ Compiler and MPI run-time environments have been set properly. You can do this using the scripts `mklvars.sh`, `compilervars.sh`, and `mpivars.sh` that are included in those distributions.
5. Run the chosen version of the benchmark.

    - The Intel AVX and Intel AVX2 optimized versions perform best with one MPI process per socket and one OpenMP* thread per core skipping simultaneous multithreading (SMT) threads: set the affinity as `KMP_AFFINITY=granularity=fine,compact,1,0`. Specifically, for a 128-node cluster with two Intel Xeon Processor E5-2697 v4 per node, run the executable as follows:

```
#> mpiexec.hydra -n
256 -ppn 2 env OMP_NUM_THREADS=18
KMP_AFFINITY=granularity=fine,compact,1,0
./bin/xhpcg_avx2 -n192
```

- The Intel Xeon Phi processor optimized version performs best with four MPI processes per processor and two threads for each processor core, with SMT turned on. Specifically, for a 128-node cluster with one Intel Xeon Phi processor 7250 per node, run the executable in this manner:

```
#> mpiexec.hydra -n
512 -ppn 2 env OMP_NUM_THREADS=34
KMP_AFFINITY=granularity=fine,compact,1,0
./bin/xhpcg_knl -n160
```

6. When the benchmark completes execution, which usually takes a few minutes, find the YAML file with official results in the current directory. The performance rating of the benchmarked system is in the last section of the file:

```
HPCG result is VALID with a GFLOP/s rating of: [GFLOP/s]
```

Choosing Best Configuration and Problem Sizes

The performance of the Intel Optimized HPCG depends on many system parameters including (but not limited to) the hardware configuration of the host and MPI implementation used. To get the best performance for a specific system configuration, choose a combination of these parameters:

- The number of MPI processes per host and OpenMPI threads per process
- Local problem size

On Intel Xeon processor-based clusters, use the Intel AVX, Intel AVX2, or Intel AVX-512 optimized version of the benchmark depending on the supported instruction set and run one MPI process per CPU socket and one OpenMP* thread per physical CPU core skipping SMT threads.

On systems based on Intel Xeon Phi processors, use the Intel AVX-512 optimized version with four MPI processes per processor. Set the number of OpenMP threads to two for each processor core, with SMT turned on. For example, on Intel Xeon Phi processor 7250 which has 68 cores, each MPI process should run 34 OpenMP threads.

For best performance, use the problem size that is large enough to better utilize available cores, but not too large, so that all tasks fit the available memory.

Intel® Math Kernel Library Language Interfaces Support
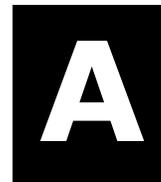
Language Interfaces Support, by Function Domain

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

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

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

Include Files

The table below lists Intel® MKL include files.

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

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

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

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

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

See Also
Language Interfaces Support, by Function Domain

B

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, the FFTW3 interface is also integrated in Intel® MKL.

Caution

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

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

C

Directory Structure in Detail

Tables in this section show contents of the Intel® Math Kernel Library (Intel® MKL) architecture-specific directories.

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

See Also
High-level Directory Structure
Using Language-Specific Interfaces with Intel® MKL

Intel® Math Kernel Library Benchmarks

Detailed Structure of the IA–32 Architecture Directories

Static Libraries in the `lib/ia32_lin` Directory

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

File	Contents	Optional Component	
		Name	**Installed by Default**
Interface Layer			
`libmkl_intel.a`	Interface library for the Intel compilers. Also use for other supported compilers that do not have a specialized Intel® MKL interface library.		
`libmkl_blas95.a`	Fortran 95 interface library for BLAS for the Intel® Fortran compiler.	Fortran 95 interfaces for BLAS and LAPACK	Yes

File	Contents	Optional Component	
		Name	**Installed by Default**
`libmkl_lapack95.a`	Fortran 95 interface library for LAPACK for the Intel Fortran compiler.	Fortran 95 interfaces for BLAS and LAPACK	Yes
`libmkl_gf.a`	Interface library for the GNU* Fortran compiler.	GNU* Compiler Collection support	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_gnu_thread.a`	OpenMP threading library for the GNU Fortran and C compilers	GNU* Compiler Collection support	Yes
`libmkl_sequential.a`	Sequential library		
Computational Layer			
`libmkl_core.a`	Kernel library for the IA-32 architecture		

Dynamic Libraries in the `lib/ia32_lin` 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.so`	Single Dynamic Library		
Interface Layer			
`libmkl_intel.so`	Interface library for the Intel compilers. Also use for other supported compilers that do not have a specialized Intel® MKL interface library.		
`libmkl_gf.so`	Interface library for the GNU Fortran compiler	GNU* Compiler Collection support	Yes
Threading Layer			

File	Contents	Optional Component	
		Name	**Installed by Default**
`libmkl_intel_thread.so`	OpenMP threading library for the Intel compilers		
`libmkl_tbb_thread.so`	Intel TBB threading library for the Intel compilers	Intel TBB threading support	Yes
`libmkl_gnu_thread.so`	OpenMP threading library for the GNU Fortran and C compilers	GNU* Compiler Collection support	Yes
`libmkl_sequential.so`	Sequential library		
Computational Layer			
`libmkl_core.so`	Library dispatcher for dynamic load of processor-specific kernel library		
`libmkl_p4.so`	Pentium® 4 processor kernel library		
`libmkl_p4m.so`	Kernel library for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) enabled processors		
`libmkl_p4m3.so`	Kernel library for Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) enabled processors		
`libmkl_avx.so`	Kernel library for Intel® Advanced Vector Extensions (Intel® AVX) enabled processors		
`libmkl_avx2.so`	Kernel library for Intel® Advanced Vector Extensions 2 (Intel® AVX2) enabled processors		
`libmkl_avx512.so`	Kernel library for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) enabled processors		

File	Contents	Optional Component	
		Name	**Installed by Default**
`libmkl_vml_p4.so`	Vector Mathematics (VM)/Vector Statistics (VS)/Data Fitting (DF) part of Pentium® 4 processor kernel		
`libmkl_vml_p4m.so`	VM/VS/DF for Intel® SSSE3 enabled processors		
`libmkl_vml_p4m2.so`	VM/VS/DF for 45nm Hi-k Intel® Core™2 and Intel Xeon® processor families		
`libmkl_vml_p4m3.so`	VM/VS/DF for Intel® SSE4.2 enabled processors		
`libmkl_vml_avx.so`	VM/VS/DF optimized for Intel® AVX enabled processors		
`libmkl_vml_avx2.so`	VM/VS/DF optimized for Intel® AVX2 enabled processors		
`libmkl_vml_avx512.so`	VM/VS/DF optimized for Intel® AVX-512 enabled processors		
`libmkl_vml_ia.so`	VM/VS/DF default kernel for newer Intel® architecture processors		
`libmkl_vml_cmpt.so`	VM/VS/DF library for conditional numerical reproducibility		
Message Catalogs			
`locale/en_US/mkl_msg.cat`	Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English		
`locale/ja_JP/mkl_msg.cat`	Catalog of Intel® MKL messages in Japanese. Available only if Intel® MKL provides Japanese localization. Please see the Release Notes for this information.		

Detailed Structure of the Intel® 64 Architecture Directories

Static Libraries in the `lib/intel64_lin` Directory

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

File	Contents	Optional Component	
		Name	**Installed by Default**
Interface Layer			
`libmkl_intel_lp64.a`	LP64 interface library for the Intel compilers. Also use for other supported compilers that do not have a specialized Intel® MKL interface library		
`libmkl_intel_ilp64.a`	ILP64 interface library for the Intel compilers. Also use for other supported compilers that do not have a specialized Intel® MKL interface library		
`libmkl_blas95_lp64.a`	Fortran 95 interface library for BLAS for the Intel® Fortran compiler. Supports the LP64 interface	Fortran 95 interfaces for BLAS and LAPACK	Yes
`libmkl_blas95_ilp64.a`	Fortran 95 interface library for BLAS for the Intel® Fortran compiler. Supports the ILP64 interface	Fortran 95 interfaces for BLAS and LAPACK	Yes
`libmkl_lapack95_lp64.a`	Fortran 95 interface library for LAPACK for the Intel® Fortran compiler. Supports the 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. Supports the ILP64 interface	Fortran 95 interfaces for BLAS and LAPACK	Yes
`libmkl_gf_lp64.a`	LP64 interface library for the GNU Fortran compilers	GNU* Compiler Collection support	Yes
`libmkl_gf_ilp64.a`	ILP64 interface library for the GNU Fortran compilers	GNU* Compiler Collection support	Yes
Threading Layer			

File	Contents	Optional Component	
		Name	Installed by Default
`libmkl_intel_thread.a`	OpenMP threading library for the Intel compilers		
`libmkl_tbb_thread.a`	Intel TBB threading library for the Intel compilers	Intel TBB threading support	Yes
`libmkl_gnu_thread.a`	OpenMP threading library for the GNU Fortran and C compilers	GNU* Compiler Collection 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 for the Intel® 64 architecture		
Cluster Libraries			
`libmkl_scalapack_lp64.a`	ScaLAPACK routine library supporting the LP64 interface	Cluster support	
`libmkl_scalapack_ilp64.a`	ScaLAPACK routine library supporting the ILP64 interface	Cluster support	
`libmkl_cdft_core.a`	Cluster version of FFT functions.	Cluster support	
`libmkl_blacs_intelmpi_lp64.a`	LP64 version of BLACS routines for Intel® MPI Library and MPICH2 or higher.	Cluster support	
`libmkl_blacs_intelmpi_ilp64.a`	ILP64 version of BLACS routines for Intel MPI Library and MPICH2 or higher.	Cluster support	
`libmkl_blacs_openmpi_lp64.a`	LP64 version of BLACS routines supporting Open MPI.	Cluster support	
`libmkl_blacs_openmpi_ilp64.a`	ILP64 version of BLACS routines supporting Open MPI.	Cluster support	
`libmkl_blacs_sgimpt_lp64.a`	LP64 version of BLACS routines supporting SGI Message-passing Interface Toolkit* (MPT).	Cluster support	

File	Contents	Optional Component	
		Name	**Installed by Default**
`libmkl_blacs_sgimpt_ilp64.a`	ILP64 version of BLACS routines supporting SGI MPT.	Cluster support	

Dynamic Libraries in the `lib/intel64_lin` 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.so`	Single Dynamic Library		
Interface Layer			
`libmkl_intel_lp64.so`	LP64 interface library for the Intel compilers. Also use for other supported compilers that do not have a specialized Intel® MKL interface library		
`libmkl_intel_ilp64.so`	ILP64 interface library for the Intel compilers. Also use for other supported compilers that do not have a specialized Intel® MKL interface library		
`libmkl_gf_lp64.so`	LP64 interface library for the GNU Fortran compilers	GNU* Compiler Collection support	Yes
`libmkl_gf_ilp64.so`	ILP64 interface library for the GNU Fortran compilers	GNU* Compiler Collection support	Yes
Threading Layer			
`libmkl_intel_thread.so`	OpenMP threading library for the Intel compilers		
`libmkl_tbb_thread.so`	Intel TBB threading library for the Intel compilers	Intel TBB threading support	Yes
`libmkl_gnu_thread.so`	OpenMP threading library for the GNU Fortran and C compilers	GNU* Compiler Collection support	Yes

File	Contents	Optional Component	
		Name	**Installed by Default**
`libmkl_pgi_thread.so`	OpenMP threading library for the PGI* compiler	PGI* Compiler support	
`libmkl_sequential.so`	Sequential library		
Computational Layer			
`libmkl_core.so`	Library dispatcher for dynamic load of processor-specific kernel		
`libmkl_def.so`	Default kernel library		
`libmkl_mc.so`	Kernel library for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) enabled processors		
`libmkl_mc3.so`	Kernel library for Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2) enabled processors		
`libmkl_avx.so`	Kernel library for Intel® Advanced Vector Extensions (Intel® AVX) enabled processors		
`libmkl_avx2.so`	Kernel library for Intel® Advanced Vector Extensions 2 (Intel® AVX2) enabled processors		
`libmkl_avx512.so`	Kernel library for dispatching Intel® Advanced Vector Extensions 512 (Intel® AVX-512) on Intel® Xeon® processors		
`libmkl_avx512_mic.so`	Kernel library for dispatching Intel® Advanced Vector Extensions 512 (Intel® AVX-512) on Intel® Xeon Phi™ processors		
`libmkl_vml_def.so`	Vector Mathematics (VM)/Vector Statistics (VS)/Data Fitting (DF) part of default kernels		

File	Contents	Optional Component	
		Name	**Installed by Default**
`libmkl_vml_mc.so`	VM/VS/DF for Intel® SSSE3 enabled processors		
`libmkl_vml_mc2.so`	VM/VS/DF for 45nm Hi-k Intel® Core™2 and Intel® Xeon® processor families		
`libmkl_vml_mc3.so`	VM/VS/DF for Intel® SSE4.2 enabled processors		
`libmkl_vml_avx.so`	VM/VS/DF optimized for Intel® AVX enabled processors		
`libmkl_vml_avx2.so`	VM/VS/DF optimized for Intel® AVX2 enabled processors		
`libmkl_vml_avx512.so`	VM/VS/DF optimized for Intel® AVX-512 on Intel® Xeon® processors		
`libmkl_vml_avx512_mic.so`	VM/VS/DF optimized for Intel® AVX-512 on Intel® Xeon Phi™ processors		
`libmkl_vml_cmpt.so`	VM/VS/DF library for conditional numerical reproducibility		
Cluster Libraries			
`libmkl_scalapack_lp64.so`	ScaLAPACK routine library supporting the LP64 interface	Cluster support	
`libmkl_scalapack_ilp64.so`	ScaLAPACK routine library supporting the ILP64 interface	Cluster support	
`libmkl_cdft_core.so`	Cluster version of FFT functions.	Cluster support	
`libmkl_blacs_intelmpi_lp64.so`	LP64 version of BLACS routines for Intel® MPI Library and MPICH2 or higher.	Cluster support	
`libmkl_blacs_intelmpi_ilp64.so`	ILP64 version of BLACS routines for Intel MPI Library and MPICH2 or higher.	Cluster support	
`libmkl_blacs_openmpi_lp64.so`	LP64 version of BLACS routines for Open MPI.	Cluster support	

File	Contents	Optional Component	
		Name	**Installed by Default**
libmkl_blacs_openmpi_ilp64.so	ILP64 version of BLACS routines for Open MPI.	Cluster support	
libmkl_blacs_sgimpt_lp64.so	LP64 version of BLACS routines for SGI MPI Toolkit* (MPT)	Cluster support	
libmkl_blacs_sgimpt_ilp64.so	ILP64 version of BLACS routines for SGI MPT.	Cluster support	
Message Catalogs			
locale/en_US/mkl_msg.cat	Catalog of Intel® Math Kernel Library (Intel® MKL) messages in English		
locale/ja_JP/mkl_msg.cat	Catalog of Intel® MKL messages in Japanese. Available only if Intel® MKL provides Japanese localization. Please see the Release Notes for this information.		

Index

A

affinity mask *54*
aligning data, example *77*
architecture support *19*

B

BLAS
    calling routines from C *64*
    Fortran 95 interface to *63*
    OpenMP* threaded routines *40*

C

C interface to LAPACK, use of *64*
C, calling LAPACK, BLAS, CBLAS from *64*
C/C++, Intel(R) MKL complex types *65*
calling
    BLAS functions from C *66*
    CBLAS interface from C *66*
    complex BLAS Level 1 function from C *66*
    complex BLAS Level 1 function from C++ *66*
    Fortran-style routines from C *64*
CBLAS interface, use of *64*
Cluster FFT
    environment variable for *92*, *94*
    linking with *84*
    managing performance of *92*, *94*
cluster software, Intel(R) MKL *84*
cluster software, linking with
    commands *84*
    linking examples *88*
Cluster Sparse Solver, linking with *84*
code examples, use of *16*
coding
    data alignment *77*
    techniques to improve performance *57*
compilation, Intel(R) MKL version-dependent *78*
compiler run-time libraries, linking with *35*
compiler-dependent function *63*
complex types in C and C++, Intel(R) MKL *65*
computation results, consistency *69*
computational libraries, linking with *34*
conditional compilation *78*
configuring Eclipse* CDT *97*
consistent results *69*
conventions, notational *11*
custom shared object
    building *36*
    composing list of functions *38*
    specifying function names *38*

D

data alignment, example *77*
denormal number, performance *58*
direct call, to Intel(R) Math Kernel Library computational kernels *55*

directory structure

directory structure
    high-level *19*
    in-detail
dispatch Intel(R) architectures, configure with an environment variable *94*
dispatch, new Intel(R) architectures, enable with an environment variable *94*

E

Eclipse* CDT
    configuring *97*
Enter index keyword *22*
environment variables
    for threading control *47*
    setting for specific architecture and programming interface *14*, *15*
    to control dispatching for Intel(R) architectures *94*
    to control threading algorithm for ?gemm *50*
    to enable dispatching of new architectures *94*
    to manage behavior of function domains *91*
    to manage behavior of Intel(R) Math Kernel Library with *91*
    to manage performance of cluster FFT *92*
examples, linking
    for cluster software *88*
    general *25*

F

FFT interface
    OpenMP* threaded problems *40*
FFTW interface support *114*
Fortran 95 interface libraries *32*
function call information, enable printing *80*

H

header files, Intel(R) MKL *112*
heterogeneity
    of Intel(R) Distribution for LINPACK* Benchmark *100*
heterogeneous cluster
    support by Intel(R) Distribution for LINPACK* Benchmark for Clusters *104*
high-bandwidth memory, use in Intel(R) Math Kernel Library *59*
HT technology, configuration tip *53*

I

ILP64 programming, support for *30*
improve performance, for matrices of small sizes *55*
include files, Intel(R) MKL *112*
information, for function call , enable printing *80*
installation, checking *13*
Intel(R) Distribution for LINPACK* Benchmark
    heterogeneity of *100*
Intel(R) Distribution for LINPACK* Benchmark for Clusters

V

Vector Mathematics
    default mode, setting with environment variable *91*
    environment variable to set default mode *91*
verbose mode, of Intel(R) MKL *80*