Intel® Math Kernel Library

Getting Started Tutorial: Using the Intel® Math Kernel Library for Matrix Multiplication

Legal Information

Legal Information

Third Party Content

Intel® Math Kernel Library (Intel® MKL) includes content from several 3rd party sources that was originally governed by the licenses referenced below:

-
- Sections on the Linear Algebra PACKage (LAPACK) routines include derivative work portions that have been copyrighted:

- Intel MKL supports LAPACK 3.5 set of computational, driver, auxiliary and utility routines under the following license:

The original versions of LAPACK from which that part of Intel MKL was derived can be obtained from http://www.netlib.org/lapack/index.html. The authors of LAPACK are E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen.

- The original versions of the Basic Linear Algebra Subprograms (BLAS) from which the respective part of Intel® MKL was derived can be obtained from http://www.netlib.org/blas/index.html.
- XBLAS is distributed under the following copyright:

- The original versions of the Basic Linear Algebra Communication Subprograms (BLACS) from which the respective part of Intel MKL was derived can be obtained from http://www.netlib.org/blacs/index.html. The authors of BLACS are Jack Dongarra and R. Clint Whaley.

- The original versions of Scalable LAPACK (ScaLAPACK) from which the respective part of Intel® MKL was derived can be obtained from http://www.netlib.org/scalapack/index.html. The authors of ScaLAPACK are L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley.
- The original versions of the Parallel Basic Linear Algebra Subprograms (PBLAS) routines from which the respective part of Intel® MKL was derived can be obtained from http://www.netlib.org/scalapack/html/pblas_qref.html.
- PARDISO (PARallel DIrect SOlver)* in Intel® MKL was originally developed by the Department of Computer Science at the University of Basel (http://www.unibas.ch). It can be obtained at http://www.pardiso-project.org.
- The Extended Eigensolver functionality is based on the Feast solver package and is distributed under the following license:

- Some Fast Fourier Transform (FFT) functions in this release of Intel® MKL have been generated by the SPIRAL software generation system (http://www.spiral.net/) under license from Carnegie Mellon University. The authors of SPIRAL are Markus Puschel, Jose Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo.
- Open MPI is distributed under the New BSD license, listed below.

Most files in this release are marked with the copyrights of the organizations who have edited them. The copyrights below are in no particular order and generally reflect members of the Open MPI core team who have contributed code to this release. The copyrights for code used under license from other parties are included in the corresponding files.

Getting Technical 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/.

Overview

Discover how to incorporate core math functions from the Intel® Math Kernel Library (Intel® MKL) to improve the performance of your application.

About This Tutorial	This tutorial demonstrates how to use Intel MKL in your applications: • Multiplying matrices using Intel MKL routines • Measuring performance of matrix multiplication • Controlling threading
Estimated Duration	10-20 minutes.
Learning Objectives	After you complete this tutorial, you should be able to: • Use Intel MKL routines for linear algebra • Compile and link your code • Measure performance using support functions • Understand the impact of threading on Intel MKL performance • Control threading for Intel MKL functions
More Resources	This tutorial uses the Fortran language, but the concepts and procedures in this tutorial apply regardless of programming language. A similar tutorial using a sample application in another programming language may be available at http://software.intel.com/en-us/articles/intel-software-product-tutorials/. This site also offers a printable version (PDF) of tutorials. In addition, you can find more resources at http://software.intel.com/en-us/articles/intel-mkl/.

Click here for more Getting Started Tutorials

Introduction to the Intel® Math Kernel Library

Use the Intel Math Kernel Library (Intel MKL) when you need to perform computations with high performance. Intel MKL offers highly-optimized and extensively threaded routines which implement many types 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-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

Operation	Routine
Linear Algebra	• BLAS • LAPACK/ScaLAPACK • PARDISO* • Iterative sparse solvers
Fast Fourier Transforms	• Multi-dimensional (up to 7D) FFTs • FFTW interfaces • Cluster FFT
Summary Statistics	• Kurtosis • Variation coefficient • Quantiles, order statistics • Min/max • Variance/covariance • ...
Data Fitting	• Splines • Interpolation • Cell search

Other Components	• Vector Math
	  • Trigono-metric
	  • Hyperbolic
	  • Exponential, Logarithmic
	  • Power/Root
	  • Rounding
	• Vector Random Number Generators
	  • Congruential
	  • Recursive
	  • Wichmann-Hill
	  • Mersenne Twister
	  • Sobol
	  • Niederreiter

Operation	Routine
	• RDRAND-based • Poisson Solvers • Optimization Solvers

Explore Basic Linear Algebra Subprograms (BLAS)

One key area is the Basic Linear Algebra Subprograms (BLAS), which perform a variety of vector and matrix operations. This tutorial uses the `dgemm` routine to demonstrate how to perform matrix multiplication as efficiently as possible.

Click here for more Getting Started Tutorials

Multiplying Matrices Using dgemm

Intel MKL provides several routines for multiplying matrices. The most widely used is the `dgemm` routine, which calculates the product of double precision matrices:

$$C \Leftarrow \alpha A * B + \beta C$$

The `dgemm` routine can perform several calculations. For example, you can perform this operation with the transpose or conjugate transpose of *A* and *B*. The complete details of capabilities of the `dgemm` routine and all of its arguments can be found in the ?gemm topic in the *Intel Math Kernel Library Developer Reference*.

Use dgemm to Multiply Matrices

This exercise demonstrates declaring variables, storing matrix values in the arrays, and calling `dgemm` to compute the product of the matrices. The arrays are used to store these matrices:

$$A = \begin{bmatrix} 1.0 & 1001.0 & 2001.0 & \cdots & 999001.0 \\ 2.0 & 1002.0 & 2002.0 & \cdots & 999002.0 \\ 3.0 & 1003.0 & 2003.0 & \cdots & 999003.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1000.0 & 2000.0 & 3000.0 & \cdots & 1000000.0 \end{bmatrix} \quad B = \begin{bmatrix} -1.0 & -1001.0 & -2001.0 & \cdots & -999001.0 \\ -2.0 & -1002.0 & -2002.0 & \cdots & -999002.0 \\ -3.0 & -1003.0 & -2003.0 & \cdots & -999003.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1000.0 & -2000.0 & -3000.0 & \cdots & -1000000.0 \end{bmatrix}$$

The one-dimensional arrays in the exercises store the matrices by placing the elements of each column in successive cells of the arrays.

NOTE
The Fortran source code for the exercises in this tutorial can be downloaded from https://software.intel.com/en-us/product-code-samples.

Although Intel MKL supports Fortran 90 and later, the exercises in this tutorial use FORTRAN 77 for compatibility with as many versions of Fortran as possible.

```
* Fortran source code is found in dgemm_example.f

      PROGRAM   MAIN

      IMPLICIT NONE

      DOUBLE PRECISION ALPHA, BETA
      INTEGER          M, K, N, I, J
      PARAMETER        (M=2000, K=200, N=1000)
      DOUBLE PRECISION A(M,K), B(K,N), C(M,N)

      PRINT *, "This example computes real matrix C=alpha*A*B+beta*C"
      PRINT *, "using Intel(R) MKL function dgemm, where A, B, and C"
      PRINT *, "are matrices and alpha and beta are double precision "
      PRINT *, "scalars"
      PRINT *, ""
```

```
      PRINT *, "Initializing data for matrix multiplication C=A*B for "
      PRINT 10, " matrix A(",M," x",K, ") and matrix B(", K," x", N, ")"
10    FORMAT(a,I5,a,I5,a,I5,a,I5,a)
      PRINT *, ""
      ALPHA = 1.0
      BETA = 0.0

      PRINT *, "Intializing matrix data"
      PRINT *, ""
      DO I = 1, M
        DO J = 1, K
          A(I,J) = (I-1) * K + J
        END DO
      END DO

      DO I = 1, K
        DO J = 1, N
          B(I,J) = -((I-1) * N + J)
        END DO
      END DO

      DO I = 1, M
        DO J = 1, N
          C(I,J) = 0.0
        END DO
      END DO

      PRINT *, "Computing matrix product using Intel(R) MKL DGEMM "
      PRINT *, "subroutine"
      CALL DGEMM('N','N',M,N,K,ALPHA,A,M,B,K,BETA,C,M)
      PRINT *, "Computations completed."
      PRINT *, ""

      PRINT *, "Top left corner of matrix A:"
      PRINT 20, ((A(I,J), J = 1,MIN(K,6)), I = 1,MIN(M,6))
      PRINT *, ""

      PRINT *, "Top left corner of matrix B:"
      PRINT 20, ((B(I,J),J = 1,MIN(N,6)), I = 1,MIN(K,6))
      PRINT *, ""

 20   FORMAT(6(F12.0,1x))

      PRINT *, "Top left corner of matrix C:"
      PRINT 30, ((C(I,J), J = 1,MIN(N,6)), I = 1,MIN(M,6))
      PRINT *, ""

 30   FORMAT(6(ES12.4,1x))

      PRINT *, "Example completed."
      STOP

      END
```

NOTE
This exercise illustrates how to call the `dgemm` routine. An actual application would make use of the result of the matrix multiplication.

This call to the `dgemm` routine multiplies the matrices:

```
CALL DGEMM('N','N',M,N,K,ALPHA,A,M,B,K,BETA,C,M)
```

The arguments provide options for how Intel MKL performs the operation. In this case:

`'N'`	Character indicating that the matrices *A* and *B* should not be transposed or conjugate transposed before multiplication.
`M, N, K`	Integers indicating the size of the matrices: • *A*: `M` rows by `K` columns • *B*: `K` rows by `N` columns • *C*: `M` rows by `N` columns
`ALPHA`	Real value used to scale the product of matrices *A* and *B*.
`A`	Array used to store matrix *A*.
`M`	Leading dimension of array `A`, or the number of elements between successive columns (for column major storage) in memory. In the case of this exercise the leading dimension is the same as the number of rows.
`B`	Array used to store matrix *B*.
`K`	Leading dimension of array `B`, or the number of elements between successive columns (for column major storage) in memory. In the case of this exercise the leading dimension is the same as the number of rows.
`BETA`	Real value used to scale matrix *C*.
`C`	Array used to store matrix *C*.
`M`	Leading dimension of array `C`, or the number of elements between successive columns (for column major storage) in memory. In the case of this exercise the leading dimension is the same as the number of rows.

Compile and Link Your Code

Intel MKL provides many options for creating code for multiple processors and operating systems, compatible with different compilers and third-party libraries, and with different interfaces. To compile and link the exercises in this tutorial with Intel® Parallel Studio XE Composer Edition, type

- Windows* OS: `ifort /Qmkl src\dgemm_example.f`
- Linux* OS, macOS*: `ifort -mkl src/dgemm_example.f`

Alternatively, you can use the supplied build scripts to build and run the executables.

- Windows* OS:

```
 build
build run_dgemm_example
```

- Linux* OS, macOS*:

```
make
make run_dgemm_example
```

For the executables in this tutorial, the build scripts are named:

Example	**Executable**
`dgemm_example.f`	`run_dgemm_example`
`dgemm_with_timing.f`	`run_dgemm_with_timing`
`matrix_multiplication.f`	`run_matrix_multiplication`
`dgemm_threading_effect_example.f`	`run_dgemm_threading_effect_example`

NOTE
This assumes that you have installed Intel MKL and set environment variables as described in https://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-2019-getting-started.

For other compilers, use the Intel MKL Link Line Advisor to generate a command line to compile and link the exercises in this tutorial: http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/.

After compiling and linking, execute the resulting executable file, named `dgemm_example.exe` on Windows* OS or `a.out` on Linux* OS and macOS*.

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 MKL Documentation for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

Intel Math Kernel Library Knowledge Base for articles describing usage of Intel MKL functionality.

for articles describing usage of Intel MKL functionality.

Click here for more Getting Started Tutorials

Measuring Performance with Intel® MKL Support Functions

Intel MKL provides functions to measure performance. This provides a way of quantifying the performance improvement resulting from using Intel MKL routines in this tutorial.

Measure Performance of dgemm

Use the `dsecnd` routine to return the elapsed CPU time in seconds.

> **NOTE**
> The quick execution of the `dgemm` routine makes it difficult to measure its speed, even for an operation on a large matrix. For this reason, the exercises perform the multiplication multiple times. You should set the value of the `LOOP_COUNT` constant so that the total execution time is about one second.

```
*       Fortran source code is found in dgemm_with_timing.f

      PRINT *, "Making the first run of matrix product using "
      PRINT *, "Intel(R) MKL DGEMM subroutine to get stable "
      PRINT *, "run time measurements"
      PRINT *, ""
      CALL DGEMM('N','N',M,N,K,ALPHA,A,M,B,K,BETA,C,M)

      PRINT *, "Measuring performance of matrix product using "
      PRINT *, "Intel(R) MKL DGEMM subroutine"
      PRINT *, ""
      S_INITIAL = DSECND()
      DO R = 1, LOOP_COUNT
        CALL DGEMM('N','N',M,N,K,ALPHA,A,M,B,K,BETA,C,M)
      END DO
      S_ELAPSED = (DSECND() - S_INITIAL) / LOOP_COUNT
      PRINT *, "== Matrix multiplication using Intel(R) MKL DGEMM =="
      PRINT 50, " == completed at ",S_ELAPSED*1000," milliseconds =="
 50   FORMAT(A,F12.5,A)
      PRINT *, ""
```

Measure Performance Without Using dgemm

In order to show the improvement resulting from using `dgemm`, perform the same measurement, but use a triply-nested loop to multiply the matrices.

```
*        Fortran source code is found in matrix_multiplication.f

      PRINT *, "Making the first run of matrix product using "
      PRINT *, "triple nested loop to get stable run time"
      PRINT *, "measurements"
      PRINT *, ""
      DO I = 1, M
        DO J = 1, N
          TEMP = 0.0
          DO L = 1, K
            TEMP = TEMP + A(I,L) * B(L,J)
```

```
          END DO
          C(I,J) = TEMP
        END DO
      END DO

      PRINT *, "Measuring performance of matrix product using "
      PRINT *, "triple nested loop"
      PRINT *, ""
      S_INITIAL = DSECND()
      DO R = 1, LOOP_COUNT
        DO I = 1, M
          DO J = 1, N
            TEMP = 0.0
            DO L = 1, K
              TEMP = TEMP + A(I,L) * B(L,J)
            END DO
            C(I,J) = TEMP
          END DO
        END DO
      END DO
      S_ELAPSED = (DSECND() - S_INITIAL) / LOOP_COUNT
      PRINT *, "== Matrix multiplication using triple nested loop =="
      PRINT 50, " == completed at ",S_ELAPSED*1000," milliseconds =="
 50   FORMAT(A,F12.5,A)
      PRINT *, ""
```

Compare the results in the first exercise using `dgemm` to the results of the second exercise without using `dgemm`.

You can find more information about measuring Intel MKL performance from the article "A simple example to measure the performance of an Intel MKL function" in the Intel Math Kernel Library Knowledge Base.

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 MKL Documentation for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

Intel Math Kernel Library Knowledge Base for articles describing usage of Intel MKL functionality.

for articles describing usage of Intel MKL functionality.

Click here for more Getting Started Tutorials

Measuring Effect of Threading on dgemm

By default, Intel MKL uses *n* threads, where *n* is the number of physical cores on the system. By restricting the number of threads and measuring the change in performance of `dgemm`, this exercise shows how threading impacts performance.

Limit the Number of Cores Used for dgemm

This exercise uses the `mkl_set_num_threads` routine to override the default number of threads, and `mkl_get_max_threads` to determine the maximum number of threads.

```
*      Fortran source code is found in dgemm_threading_effect_example.f

      PRINT *, "Finding max number of threads Intel(R) MKL can use for"
      PRINT *, "parallel runs"
      PRINT *, ""
      MAX_THREADS = MKL_GET_MAX_THREADS()

      PRINT 20," Running Intel(R) MKL from 1 to ",MAX_THREADS," threads"
 20   FORMAT(A,I2,A)
      PRINT *, ""
      DO L = 1, MAX_THREADS
        DO I = 1, M
          DO J = 1, N
            C(I,J) = 0.0
          ENDDO
        ENDDO

        PRINT 30, " Requesting Intel(R) MKL to use ",L," thread(s)"
 30     FORMAT(A,I2,A)
        CALL MKL_SET_NUM_THREADS(L)

        PRINT *, "Making the first run of matrix product using "
        PRINT *, "Intel(R) MKL DGEMM subroutine to get stable "
        PRINT *, "run time measurements"
        PRINT *, ""
        CALL DGEMM('N','N',M,N,K,ALPHA,A,M,B,K,BETA,C,M)

        PRINT *, "Measuring performance of matrix product using "
        PRINT 40, " Intel(R) MKL DGEMM subroutine on ",L," thread(s)"
 40     FORMAT(A,I2,A)
        PRINT *, ""
        S_INITIAL = DSECND()
        DO R = 1, LOOP_COUNT
          CALL DGEMM('N','N',M,N,K,ALPHA,A,M,B,K,BETA,C,M)
        END DO
        S_ELAPSED = (DSECND() - S_INITIAL) / LOOP_COUNT

        PRINT *, "== Matrix multiplication using Intel(R) MKL DGEMM =="
        PRINT 50, " == completed at ",S_ELAPSED*1000," milliseconds =="
        PRINT 60, " == using ",L," thread(s) =="
 50     FORMAT(A,F12.5,A)
```

```
60     FORMAT(A,I2,A)
       PRINT *, ""
     END DO
```

Examine the results shown and notice that time to multiply the matrices decreases as the number of threads increases. If you try to run this exercise with more than the number of threads returned by `mkl_get_max_threads`, you might see performance degrade when you use more threads than physical cores.

> **NOTE**
> You can see specific performance results for `dgemm` at the Details tab at http://software.intel.com/en-us/articles/intel-mkl.

See Also

Intel MKL Documentation for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

Intel Math Kernel Library Knowledge Base for articles describing usage of Intel MKL functionality.

for articles describing usage of Intel MKL functionality.

Click here for more Getting Started Tutorials

Other Areas to Explore

The exercises so far have given the basic ideas needed to get started with Intel MKL, but there are plenty of other areas to explore. The following are some controls, interfaces, and topics which you might find worth investigating further.

Support functions	The second exercise shows how to use the timing functions and the third exercise shows the use of threading control functions. Acquaint yourself with other support functions by referring to the "Support functions" chapter of the *Intel MKL Developer Reference*: • Support functions for Conditional Numerical Reproducibility (CNR) These functions provide the means to balance reproducibility with performance in certain conditions. • Memory functions These functions provide support for allocating and freeing memory. The allocation functions allow proper alignment of memory to ensure reproducibility when used together with CBWR functions. • Error handling functions The `xerbla` function is used by BLAS, LAPACK, VML, and VSL to report errors.
Linking and interfaces	• The ILP64 interface Most users call the interface of Intel MKL that takes 32-bit integers for size parameters, but increased memory and also some legacy code requires 64-bit integers. Read more about the ILP64 interface and the libraries and functions supporting it in the *Intel MKL Developer Guide*. • Single Dynamic Library (SDL) linking model Intel MKL has two ways to link to dynamic libraries. The newest of these models is the best option for those calling Intel MKL from managed runtime libraries and is easy to link, but requires some functions calls to use non-default interfaces (for example, ILP64). See the *Intel MKL Developer Guide* for more information on Intel MKL linking models.
Miscellaneous	• Environment variables Many controls in Intel MKL have both environment variables and functional versions. In all cases the function overrides the behavior of the environment variable. If you do not want the behavior to change based on an environment variable in a particular case, use the function call to ensure the desired setting. See the *Intel MKL Developer Guide* for descriptions of the environment variables used by Intel MKL.

See Also

Intel MKL Documentation for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

for additional Intel MKL documentation, including the *Intel MKL Developer Reference* and the *Intel MKL Developer Guide*.

Intel Math Kernel Library Knowledge Base for articles describing usage of Intel MKL functionality.

for articles describing usage of Intel MKL functionality.

Click here for more Getting Started Tutorials