Intel® Math Kernel Library

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

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENCE IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/

Intel, Intel Atom, and Intel Core are trademarks of Intel Corporation in the U.S. and/or other countries.

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

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

Portions Copyright © 2001, Hewlett-Packard Development Company, L.P.

Copyright © 2012, Intel Corporation. All rights reserved.
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/.  


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

### 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
  - Trigonometric
  - Hyperbolic
  - Exponential, Logarithmic
  - Power/Root
  - Rounding
- Vector Random Number Generators
  - Conguential
  - Recursive
  - Wichmann-Hill
  - Mersenne Twister
  - Sobol
  - Neiderreiter
  - RDRAND-based
- Poisson Solvers
- Optimization Solvers

### Exploring 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.
Multiplying Matrices Using \texttt{dgemm}

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

\[ \mathbf{C} \leftarrow \alpha \mathbf{A} \mathbf{B} + \beta \mathbf{C} \]

The \texttt{dgemm} routine can perform several calculations. For example, you can perform this operation with the transpose or conjugate transpose of \( \mathbf{A} \) and \( \mathbf{B} \). The complete details of capabilities of the \texttt{dgemm} routine and all of its arguments can be found in the \texttt{?gemm} topic in the Intel Math Kernel Library Reference Manual.

Use \texttt{dgemm} to Multiply Matrices

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

\[
\begin{bmatrix}
1.0 & 1001.0 & 2001.0 & \cdots & 999001.0 \\
2.0 & 1002.0 & 2002.0 & \cdots & 999002.0 \\
3.0 & 1003.0 & 2003.0 & \cdots & 999003.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1000.0 & 2000.0 & 3000.0 & \cdots & 1000000.0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-1.0 & -1001.0 & -2001.0 & \cdots & -999001.0 \\
-2.0 & -1002.0 & -2002.0 & \cdots & -999002.0 \\
-3.0 & -1003.0 & -2003.0 & \cdots & -999003.0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
-1000.0 & -2000.0 & -3000.0 & \cdots & -1000000.0 \\
\end{bmatrix}
\]

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 is found in `<install-dir>\Documentation\en-US\mkl\tutorials\mkl_mm\mmx_f` (Windows* OS), or `<install-dir>/Documentation/en-US/mkl/tutorials/mkl_mmx_f` (Linux* OS/OS X*).

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 \texttt{dgemm_example.f}

---

```
PROGRAM MAIN
IMPLICIT NONE

DOUBLE PRECISION ALPHA, BETA
INTEGER M, P, N, L, J
PARAMETER (M=2000, P=200, N=1000)
DOUBLE PRECISION A(M,P), B(P,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",P," ) and matrix B("P"," x", N," )"
FORMAT(a,M,a,15,a,15,a,15,a,15,a,15,a,15,a)
PRINT *, ""
ALPHA = 1.0
BETA = 0.0
```

---
PRINT *, "Intializing matrix data"
PRINT *, ""
DO I = 1, M
  DO J = 1, P
    A(I,J) = (I-1) * P + J
  END DO
END DO
DO I = 1, P
  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"
PRINT *, "Computations completed."
PRINT *, ""
PRINT *, "Top left corner of matrix A:
PRINT 20, ((A(I,J), J = 1,MIN(P,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(P,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:


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, P Integers indicating the size of the matrices:

- A: M rows by P columns
- B: P rows by N columns
- C: M rows by N columns
**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® Composer XE, type

- **Windows* OS:** `ifort /Qmkl dgemm_example.f`
- **Linux* OS, OS X*: `ifort -mkl dgemm_example.f`

**NOTE** This assumes that you have installed Intel MKL and set environment variables as described in [Intel MKL 110 Getting Started](http://software.intel.com/en-us/articles/intel-mkl-110-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: [Intel MKL Link Line Advisor](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 OS X*.

**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 Knowledge Base for articles describing usage of Intel MKL functionality.
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
PRINT *, "Making the first run of matrix product using "
PRINT *, "Intel(R) MKL DGEMM subroutine to get stable run time measurements"
PRINT *, ""
PRINT *, "Measuring performance of matrix product using "
PRINT *, "Intel(R) MKL DGEMM subroutine"
PRINT *, ""
S_INITIAL = DSECND()
DO R = 1, LOOP_COUNT
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
PRINT *, "Making the first run of matrix product using "
PRINT *, "triple nested loop to get stable run time measurements"
PRINT *, ""
DO I = 1, M
   DO J = 1, N
      TEMP = 0.0
      DO K = 1, P
         TEMP = TEMP + A(I,K) * B(K,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 K = 1, P  
        TEMP = TEMP + A(I,K) * B(K,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 \texttt{dgemm} to the results of the second exercise without using \texttt{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.

\textbf{See Also}


Intel Math Kernel Library Knowledge Base for articles describing usage of Intel MKL functionality.
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
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 K = 1, MAX_THREADS
   DO I = 1, M
      DO J = 1, N
         C(I,J) = 0.0
      END DO
   ENDDO
   PRINT 30, " Requesting Intel(R) MKL to use ",K," thread(s)"
30     FORMAT(A,I2,A)
      CALL MKL_SET_NUM_THREADS(K)
   PRINT *, "Making the first run of matrix product using 
   PRINT *, "Intel(R) MKL DGEMM subroutine to get stable 
   PRINT *, "run time measurements"
   PRINT *, ""
      PRINT *, "Measuring performance of matrix product using 
      PRINT *, "Intel(R) MKL DGEMM subroutine on ",K," thread(s)"
40 FORMAT(A,I2,A)
      PRINT *, ""
      S_INITIAL = DSECND()
      DO R = 1, LOOP_COUNT
      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 *, "== using ",K," thread(s) =="
      PRINT 60, "== using ",K," thread(s) =="
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 \texttt{dgemm} at the Details tab at http://software.intel.com/en-us/articles/intel-mkl.

**See Also**


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

- **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 User's 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 User's 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 User's Guide for descriptions of the environment variables used by Intel MKL.

See Also


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

M
main cover 3