Intel® Math Kernel Library

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

Document Number: 327255-005US

Legal Information

Legal Information

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

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 NEGLIGENT 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

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, Flexpipe, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel CoFluent, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel Xeon Phi, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Pentium, Pentium Inside, Puma, skoool, the skoool logo, SMARTi, Sound Mark, Stay With It, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS 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.

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:

- Portions© Copyright 2001 Hewlett-Packard Development Company, L.P.
- Sections on the Linear Algebra PACKage (LAPACK) routines include derivative work portions that have been copyrighted:

  © 1991, 1992, and 1998 by The Numerical Algorithms Group, Ltd.
- Intel MKL supports LAPACK 3.5 set of computational, driver, auxiliary and utility routines under the following license:

  Copyright © 1992-2011 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.

  Copyright © 2000-2011 The University of California Berkeley. All rights reserved.

  Copyright © 2006-2012 The University of Colorado Denver. All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
  - Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

  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.
- 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:

  Copyright © 2009, The Regents of the University of Massachusetts, Amherst.
  Developed by E. Polizzi
  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
- 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.

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 C 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	**Fast Fourier Transforms**	**Summary Statistics**	**Data Fitting**	**Other Components**

• BLAS
• LAPACK/ ScaLAPACK
• PARDISO*
• Iterative sparse solvers
	• Multi-dimensional (up to 7D) FFTs
	• FFTW interfaces
	• Cluster FFT
		• Kurtosis
		• Variation coefficient
		• Quantiles, order statistics
		• Min/max
		• Variance/ covariance
		• ...
			• Splines
			• Interpolation
			• Cell search
				• Vector Math
				  • Trigono-metric
				  • 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 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 Reference Manual*.

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:

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

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 C source code for the exercises in this tutorial is found in `<install-dir>\Samples\en-US\mkl\tutorials.zip` (Windows* OS), or `<install-dir>/Samples/en-US/mkl/tutorials.zip` (Linux* OS/OS X*).
>
> After you unzip the `tutorials.zip` file, the Fortran source code can be found in the `mkl_mmx_f` directory, and the C source code can be found in the `mkl_mmx_c` directory.

```
/* C source code is found in dgemm_example.c */

#define min(x,y) (((x) < (y)) ? (x) : (y))

#include <stdio.h>
#include <stdlib.h>
#include "mkl.h"

int main()
{
    double *A, *B, *C;
    int m, n, k, i, j;
    double alpha, beta;

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

    m = 2000, k = 200, n = 1000;
```

```
printf (" Initializing data for matrix multiplication C=A*B for matrix \n"
        " A(%ix%i) and matrix B(%ix%i)\n\n", m, k, k, n);
alpha = 1.0; beta = 0.0;

printf (" Allocating memory for matrices aligned on 64-byte boundary for better \n"
        " performance \n\n");
A = (double *)mkl_malloc( m*k*sizeof( double ), 64 );
B = (double *)mkl_malloc( k*n*sizeof( double ), 64 );
C = (double *)mkl_malloc( m*n*sizeof( double ), 64 );
if (A == NULL || B == NULL || C == NULL) {
  printf( "\n ERROR: Can't allocate memory for matrices. Aborting... \n\n");
  mkl_free(A);
  mkl_free(B);
  mkl_free(C);
  return 1;
}

printf (" Intializing matrix data \n\n");
for (i = 0; i < (m*k); i++) {
    A[i] = (double)(i+1);
}

for (i = 0; i < (k*n); i++) {
    B[i] = (double)(-i-1);
}

for (i = 0; i < (m*n); i++) {
    C[i] = 0.0;
}

printf (" Computing matrix product using Intel(R) MKL dgemm function via CBLAS interface \n\n");
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            m, n, k, alpha, A, k, B, n, beta, C, n);
printf ("\n Computations completed.\n\n");

printf (" Top left corner of matrix A: \n");
for (i=0; i<min(m,6); i++) {
  for (j=0; j<min(k,6); j++) {
    printf ("%12.0f", A[j+i*k]);
  }
  printf ("\n");
}

printf ("\n Top left corner of matrix B: \n");
for (i=0; i<min(k,6); i++) {
  for (j=0; j<min(n,6); j++) {
    printf ("%12.0f", B[j+i*n]);
  }
  printf ("\n");
}

printf ("\n Top left corner of matrix C: \n");
for (i=0; i<min(m,6); i++) {
  for (j=0; j<min(n,6); j++) {
    printf ("%12.5G", C[j+i*n]);
  }
  printf ("\n");
}

printf ("\n Deallocating memory \n\n");
```

```
    mkl_free(A);
    mkl_free(B);
    mkl_free(C);

    printf (" Example completed. \n\n");
    return 0;
}
```

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:

```
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
           m, n, k, alpha, A, k, B, n, beta, C, n);
```

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

`CblasRowMajor`	Indicates that the matrices are stored in row major order, with the elements of each row of the matrix stored contiguously as shown in the figure above.
`CblasNoTrans`	Enumeration type 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*.
`k`	Leading dimension of array `A`, or the number of elements between successive rows (for row major storage) in memory. In the case of this exercise the leading dimension is the same as the number of columns.
`B`	Array used to store matrix *B*.
`n`	Leading dimension of array `B`, or the number of elements between successive rows (for row major storage) in memory. In the case of this exercise the leading dimension is the same as the number of columns.
`beta`	Real value used to scale matrix *C*.
`C`	Array used to store matrix *C*.
`n`	Leading dimension of array `C`, or the number of elements between successive rows (for row major storage) in memory. In the case of this exercise the leading dimension is the same as the number of 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® Parallel Studio XE Composer Edition, type

- Windows* OS: `icl /Qmkl dgemm_example.c`
- Linux* OS, OS X*: `icc -mkl dgemm_example.c`

> **NOTE**
> This assumes that you have installed Intel MKL and set environment variables as described in http://software.intel.com/en-us/articles/intel-mkl-112-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 OS X*.

See Also

Intel MKL Documentation for additional Intel MKL documentation, including the *Intel MKL Reference Manual* and the *Intel MKL User's Guide*.

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

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.

```
/* C source code is found in dgemm_with_timing.c */

    printf (" Making the first run of matrix product using Intel(R) MKL dgemm function \n"
            " via CBLAS interface to get stable run time measurements \n\n");
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                m, n, k, alpha, A, k, B, n, beta, C, n);

    printf (" Measuring performance of matrix product using Intel(R) MKL dgemm function \n"
            " via CBLAS interface \n\n");
    s_initial = dsecnd();
    for (r = 0; r < LOOP_COUNT; r++) {
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                    m, n, k, alpha, A, k, B, n, beta, C, n);
    }
    s_elapsed = (dsecnd() - s_initial) / LOOP_COUNT;

    printf (" == Matrix multiplication using Intel(R) MKL dgemm completed == \n"
            " == at %.5f milliseconds == \n\n", (s_elapsed * 1000));
```

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.

```
/* C source code is found in matrix_multiplication.c */

    printf (" Making the first run of matrix product using triple nested loop\n"
            " to get stable run time measurements \n\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            sum = 0.0;
            for (l = 0; l < k; l++)
                sum += A[k*i+l] * B[n*l+j];
            C[n*i+j] = sum;
        }
    }

    printf (" Measuring performance of matrix product using triple nested loop \n\n");
    s_initial = dsecnd();
    for (r = 0; r < LOOP_COUNT; r++) {
```

```
        for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++) {
                sum = 0.0;
                for (l = 0; l < k; l++)
                    sum += A[k*i+l] * B[n*l+j];
                C[n*i+j] = sum;
            }
        }
    }
    s_elapsed = (dsecnd() - s_initial) / LOOP_COUNT;

    printf (" == Matrix multiplication using triple nested loop completed == \n"
            " == at %.5f milliseconds == \n\n", (s_elapsed * 1000));
```

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.

See Also

Intel MKL Documentation for additional Intel MKL documentation, including the *Intel MKL Reference Manual* and the *Intel MKL User's Guide*.

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.

```
/* C source code is found in dgemm_threading_effect_example.c */

    printf (" Finding max number of threads Intel(R) MKL can use for parallel runs \n\n");
    max_threads = mkl_get_max_threads();

    printf (" Running Intel(R) MKL from 1 to %i threads \n\n", max_threads);
    for (i = 1; i <= max_threads; i++) {
        for (j = 0; j < (m*n); j++)
            C[j] = 0.0;

        printf (" Requesting Intel(R) MKL to use %i thread(s) \n\n", i);
        mkl_set_num_threads(i);

        printf (" Making the first run of matrix product using Intel(R) MKL dgemm function \n"
                " via CBLAS interface to get stable run time measurements \n\n");
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                    m, n, k, alpha, A, k, B, n, beta, C, n);

        printf (" Measuring performance of matrix product using Intel(R) MKL dgemm function \n"
                " via CBLAS interface on %i thread(s) \n\n", i);
        s_initial = dsecnd();
        for (r = 0; r < LOOP_COUNT; r++) {
            cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                        m, n, k, alpha, A, k, B, n, beta, C, n);
        }
        s_elapsed = (dsecnd() - s_initial) / LOOP_COUNT;

        printf (" == Matrix multiplication using Intel(R) MKL dgemm completed ==\n"
                " == at %.5f milliseconds using %d thread(s) ==\n\n", (s_elapsed * 1000), i);
    }
```

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 Reference Manual* and the *Intel MKL User's Guide*.

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 MKL Documentation for additional Intel MKL documentation, including the *Intel MKL Reference Manual* and the *Intel MKL User's Guide*.

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