

# Intel® Math Kernel Library (Intel® MKL) Lab:

## Comparing the Performance of Compiled Code to Intel MKL BLAS Libraries

---

### Disclaimers

The information contained in this document is provided for informational purposes only and represents the current view of Intel Corporation ("Intel") and its contributors ("Contributors") on the date of publication. Intel and the Contributors make no commitment to update the information contained in this document, and Intel reserves the right to make changes at any time, without notice.

DISCLAIMER. THIS DOCUMENT IS PROVIDED "AS IS." NEITHER INTEL, NOR THE CONTRIBUTORS MAKE ANY REPRESENTATIONS OF ANY KIND WITH RESPECT TO PRODUCTS REFERENCED HEREIN, WHETHER SUCH PRODUCTS ARE THOSE OF INTEL, THE CONTRIBUTORS, OR THIRD PARTIES. INTEL, AND ITS CONTRIBUTORS EXPRESSLY DISCLAIM ANY AND ALL WARRANTIES, IMPLIED OR EXPRESS, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, NON-INFRINGEMENT, AND ANY WARRANTY ARISING OUT OF THE INFORMATION CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION, ANY PRODUCTS, SPECIFICATIONS, OR OTHER MATERIALS REFERENCED HEREIN. INTEL, AND ITS CONTRIBUTORS DO NOT WARRANT THAT THIS DOCUMENT IS FREE FROM ERRORS, OR THAT ANY PRODUCTS OR OTHER TECHNOLOGY DEVELOPED IN CONFORMANCE WITH THIS DOCUMENT WILL PERFORM IN THE INTENDED MANNER, OR WILL BE FREE FROM INFRINGEMENT OF THIRD PARTY PROPRIETARY RIGHTS, AND INTEL, AND ITS CONTRIBUTORS DISCLAIM ALL LIABILITY THEREFOR.

INTEL, AND ITS CONTRIBUTORS DO NOT WARRANT THAT ANY PRODUCT REFERENCED HEREIN OR ANY PRODUCT OR TECHNOLOGY DEVELOPED IN RELIANCE UPON THIS DOCUMENT, IN WHOLE OR IN PART, WILL BE SUFFICIENT, ACCURATE, RELIABLE, COMPLETE, FREE FROM DEFECTS OR SAFE FOR ITS INTENDED PURPOSE, AND HEREBY DISCLAIM ALL LIABILITIES THEREFOR. ANY PERSON MAKING, USING OR SELLING SUCH PRODUCT OR TECHNOLOGY DOES SO AT HIS OR HER OWN RISK.

Licenses may be required. Intel, its contributors and others may have patents or pending patent applications, trademarks, copyrights or other intellectual proprietary rights covering subject matter contained or described in this document. No license, express, implied, by estoppels or otherwise, to any intellectual property rights of Intel or any other party is granted herein. It is your responsibility to seek licenses for such intellectual property rights from Intel and others where appropriate.

Limited License Grant. Intel hereby grants you a limited copyright license to copy this document for your use and internal distribution only. You may not distribute this document externally, in whole or in part, to any other person or entity.

LIMITED LIABILITY. IN NO EVENT SHALL INTEL, OR ITS CONTRIBUTORS HAVE ANY LIABILITY TO YOU OR TO ANY OTHER THIRD PARTY, FOR ANY LOST PROFITS, LOST DATA, LOSS OF USE OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF THIS DOCUMENT OR RELIANCE UPON THE INFORMATION CONTAINED HEREIN, UNDER ANY CAUSE OF ACTION OR THEORY OF LIABILITY, AND IRRESPECTIVE OF WHETHER INTEL, OR ANY CONTRIBUTOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit <http://www.intel.com/performance/resources/limits.htm>.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

Copyright © Intel Corporation, 2015. All rights reserved.

Intel, the Intel logo, Itanium and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

# Intel® Math Kernel Library (Intel® MKL) Lab

---

## Introduction

This lab shows you how to use the Intel® Math Kernel Library (Intel® MKL) as a way to maintain portable source that can take maximum advantage of the performance related architectural features of Intel® processors. You begin with compiling and optimizing source code for matrix multiply, and then progressively modify the source to use different Intel MKL routines.

You compile and run a matrix multiply program four ways: as source code, calling the BLAS function DDOT (innermost loop replaced by library call), DGEMV (innermost two loops), and DGEMM (all source replace by library). For the final step, the second processor is invoked to enable the library workload to run in parallel. Timings are taken at each step, for performance comparison.

## Objectives

At the successful completion of these lab activities, you will be able to:

- Be familiar with Intel MKL.
- Invoke the multithreading capability of Intel MKL and identify appropriate uses of these functions.

(This lab concentrates on the horse power of linear algebra, namely the matrix multiply operation. Since this operation is at the core of all optimization of linear algebra routines, the GEMM family of Intel MKL is highly optimized for Intel® architectures.)

## Equipment List

### Hardware

- Intel® 64 based system
  - 1 GB RAM (recommended)
  - 5 GB hard drive space

### Software

- Microsoft Windows\* 7, Windows XP\* Professional or Red Hat Linux\*
- Microsoft Visual Studio\* 2008 or 2010 for Windows
- Intel® compilers

## Time Required

One hour.

## Lab Activity 1: Build and Run a “Roll Your Own” Matrix Multiply Routine

---

1. Open the file `mkl-lab1-dgemm.sln` using Microsoft Visual Studio 2008 or 2010 version on Windows. The solution file is being developed with Microsoft Visual Studio 2010 version. If you are running it on Linux, please edit `mkl_lab.c` using your favorite editor.
2. Find the routine

```
void roll_your_own_multiply(/*[in]*/double* a,
                           /*[in]*/double* b,
                           /*[out]*/double* c,
                           /*[in]*/int N)
```

and implement it. This routine takes in two matrices, `a` and `b`, multiplies them and puts the result in `c`. Matrices `a` and `b` are not modified. This should be a straightforward, three-nested loop routine.

3. Build and test to verify your result.  
Make sure you understand the operation  $a*b=c$ . **(This activity takes about 10 minutes through this step to complete.)**
4. Make sure that the command line argument is always less than 7 if you want the result printed to verify the correctness of your implementation. This is because this program will print out the value of the matrices if the size of the problem uses matrices of order 1, 2, 3, 4, 5 or 6.
5. Compile and link using the Intel compilers.  
Warnings about unused variables and such might appear. These are an artifact of the way the lab is constructed. You should understand them, but do not correct for them or later builds will not work.
6. Execute and verify that the results in matrix `c` are correct (remember how you multiply matrices  $ab=c$  by hand?).
7. Execute your program by passing 500 as the command line arguments and then execute with argument 1000 and record your numbers:

N	Time
500	-----
1000	-----

8. Optimize using the best compiler switches that are you familiar with that are appropriate for this platform. Please refer compiler documentation for details on various compiler optimization switches. Do not try to modify the program. You do that in another lab activity. Record your times:

Optimized		
N		Time
500		-----
1000		-----

## Lab Activity 2: Replace the Innermost Loop with a Library Function Call

---

1. Edit the file mkl\_lab.cpp
2. Find the routine

```
void Ddot_Multiply(/*[in]*/double* a,
                  /*[in]*/double* b,
                  /*[out]*/double* c,
                  /*[in]*/ int N)
```

3. Cut and paste the implementation of the previous activity. Note that the inner most loop computes the dot product of two vectors.
4. Replace the most inner loop by a call to the Intel MKL Fortran routine DDOT or the Intel MKL CBLAS routine cblas\_ddot(). Also, ensure that all the required header files are included. **(This activity takes about 10 minutes through this step to complete.)**
5. Go to the function main and locate the commented code block that calls DdotMultiply and uncomment it.
6. To rebuild your program on Windows using Visual Studio, Select from Project Properties->Configuration Properties->Intel Performance Libraries->Use MKL and select value Parallel. This will set the include files and library paths to Intel MKL and also select the right Intel MKL libraries for linking. In Linux you must link to libiomp5 (or use -openmp compiler flag) and the pthread library. Rebuild your application.

On Linux, modify the makefile-linux to point to the MKL libraries editing the paths specified and run

```
$make -f makfile-linux
```

Please refer Intel® MKL Linking advisor tool for choosing which libraries to link from below url

<http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/>

Also, refer the article on how to use MKL in Visual Studio from the Knowledge Base

---

<http://software.intel.com/en-us/articles/intel-math-kernel-library-intel-mkl-for-windows-build-intel-mkl-program-with-intel-fortran-in-microsoft-visual-studio/>

Verify that the result is correct.

- Execute your program with 500 and 1000 passed as command line parameters. Record your times for both the processor-specific and default libraries:

DDOT Version

N	Time (default)	Time (processor-specific)
500	-----	-----
1000	-----	-----

*Is the time for DDOT less than or greater than the "roll your own" version? Why?*

*What scenario would change this result?*

## Lab Activity 3: Replace the Innermost Two Loops with a Library Function Call

---

- Edit the file `mkl_lab.cpp`. In this lab activity, you change the two inner loops of your first implementation `roll_your_own_multiply()` by a call to the Intel MKL routine `DGEMV`. `DGEMV` multiplies a vector by a matrix to give a vector.
- Find the routine

```
void Dgemv_multiply(/*[in]*/double* a,
                  /*[in]*/double* b,
                  /*[out]*/double* c,
                  /*[in]*/ int N)
```

- Cut and paste the implementation of Activity 1.  
Note that the two inner loops compute the matrix multiply of a column vector of `b` by the matrix `a`. So the `j`th column vector of matrix `c` can be assigned the result of the `j`th column vector of `b`, multiplied by the matrix `a`.
  - Replace the two inner most loops by a call to the CBLAS equivalent `cblas_dgemv`. The same remarks apply here as in Activity 2.
  - Uncomment the code block that calls `Dgemv_multiply` in the main function. **(This activity takes about 10 minutes through this step to complete.)**
  - Rebuild your program in the same manner as Activity 2.
  - Verify that the result is correct.
-

- Execute your program with 500 and 1000 passed as command line parameters. Record your times:

DGEMV Version

N	Time (default library)	Time (processor-specific)
500	-----	-----
1000	-----	-----

*How does this compare with the previous runs?*

*Is this result consistent with your previous conclusions?*

*Again, what conditions would change this result?*

## Lab Activity 4: Replace All the Matrix Multiply Source Code with a Library Function Call

---

- Edit the file mkl\_lab.cpp. In this lab activity, you change all three loops of your first implementation roll\_your\_own\_multiply() by a call to Intel MKL DGEMM routine which is a matrix matrix product.
- Find the routine

```
void Dgemm_multiply(/*[in]*/double* a,
                   /*[in]*/double* b,
                   /*[out]*/double* c,
                   (/*[in]*/ int N)
```

- Make a call to Intel MKL Fortran routine DGEMM or the CBLAS equivalent, cblas\_dgemm. The same remarks apply here as in Activities 2 and 3.
- Find the code block in the main function that calls Dgemm\_multiply and uncomment it. **(This activity takes about 10 minutes through this step to complete.)**
- Rebuild your program in the same manner as Activities 2 and 3.
- Verify that the result is correct.
- Execute your program with 500 and 1000 passed as command line parameters. Record your times:

DGEMM Version

N	Time (default library)	Time (processor-specific)
500	-----	-----
1000	-----	-----

---

*How does this compare with the previous runs?*

*Is this result consistent with your previous conclusions?*

*Again, what conditions would change this result?*

8. Open a command window in the “release” directory and set the number of threads available to DGEMM by setting the environment variable OMP\_NUM\_THREADS.

Time the results for each of the following settings:

set MKL\_NUM\_THREADS=1 \_\_\_\_\_

set MKL\_NUM\_THREADS=2 \_\_\_\_\_

set MKL\_NUM\_THREADS=3 \_\_\_\_\_

set MKL\_NUM\_THREADS=4 \_\_\_\_\_

*How are the various timings to be explained?*

---

END OF LAB

---

# Solutions

---

## Activity 1

//Brute force way of matrix multiply

```
void roll_your_own_multiply(double* a, double* b, double* c, int N)
{
    int i, j, k;
    for(i=0; i<N; i++) {
        for(j=0; j<N; j++) {
            c[N*i+j] = 0.;
            for(k=0; k<N; k++) {
                c[N*i+j] += a[N*i+k] * b[N*k+j];
            }
        }
    }
}
```

## Activity 2

//The ddot way to matrix multiply

```
void Ddot_Multiply(double* a, double* b, double* c, int N)
{
    int i, j;
    int incx = 1;
    int incy = N;
    for (i = 0; i < N; i++) {
        for (j=0; j<N; j++) {
            c[N*i+j] = blas_ddot(N, &a[N*i], incx, &b[j], incy);
        }
    }
}
```

## Activity 3

//DGEMV way of matrix multiply

```
void Dgemv_multiply(double* a, double* b, double* c, int N)
{
    int i;
    double alpha = 1.0, beta = 0.;
    int incx = 1;
    int incy = N;
    for (i = 0; i < N; i++) {
        cblas_dgemv(CblasRowMajor, CblasNoTrans, N, N, alpha, a, N, &b[i], N, beta,
        &c[i], N);
    }
}
```

## Activity 4

//DGEMM way. The PREFERRED way, especially for large matrices

```
void Dgemm_multiply(double* a, double* b, double* c, int N)
{
    double alpha = 1.0, beta = 0.;
    int incx = 1;
    int incy = N;
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, alpha, b, N, a, N, beta, c, N);
}
```