Using Intel® Math Kernel Library with MathWorks* MATLAB* on Intel® Xeon Phi™ Coprocessor System

Overview
This guide is intended to help developers use the latest version of Intel® Math Kernel Library (Intel® MKL) with MathWorks® MATLAB® on Intel® Xeon Phi™ Coprocessor System.

Intel MKL is a computational math library designed to accelerate application performance and reduce development time. It includes highly optimized and threaded dense and sparse Linear Algebra routines, Fast Fourier transforms (FFT) routines, Vector Math routines, and Statistical functions for Intel processors and coprocessors.

MATLAB is an interactive software program that performs mathematical computations and visualization. Internally MATLAB uses Intel MKL Basic Linear Algebra Subroutines (BLAS) and Linear Algebra package (LAPACK) routines to perform the underlying computations when running on Intel processors.

Intel MKL now includes a new Automatic Offload (AO) feature that enables computationally intensive Intel MKL functions to offload partial workload to attached Intel Xeon Phi coprocessors automatically and transparently.

As a result, MATLAB performance can benefit from Intel Xeon Phi coprocessors via the Intel MKL AO feature when problem sizes are large enough to amortize the cost of transferring data to the coprocessors. The article describes how to enable Intel MKL AO when Intel Xeon Phi coprocessors are present within a MATLAB computing environment.

Prerequisite
Prior to getting started, obtain access to the following software and hardware:


3. **An Intel Xeon Phi Coprocessor Development System** as described at https://software.intel.com/en-us/mic-developer

The 64bit version of Intel MKL and MATLAB should be installed at least on the development system.

This article was created based on MATLAB R2014a and Intel MKL for Windows* 11.1 update 1 and update 2 on the system:

Host machine: Intel® Xeon® CPU E5-2697 v2, 2 Twelve-Core CPUs (30MB LLC, 2.7GHz), 128GB of RAM; OS: Windows Server 2008 R2 Enterprise
Coprocessors: 2 Intel® Xeon Phi™ Coprocessors 7120A, each with 61 cores (30.5MB total cache, 1.2GHz), 16GB GDDR5 Memory
Software: Intel® Math Kernel Library (Intel® MKL) 11.1 update 1 and update 2, Intel® Manycore Platform Software Stack (MPSS) 3.2.27270.1

* 11.1 update 1 was upgraded to update 2 when the article was drafted, so two versions were tested in the article.

**Steps**

**Step 1: Determine which version of Intel MKL is used within MATLAB** via the MATLAB “version” command.

Open a MATLAB Command Windows, enter command:

```matlab
>> version -blas
ans =
    Intel(R) Math Kernel Library Version 11.0.5 Product Build 20130612 for Intel(R) 64 architecture applications

>> version -lapack
ans =
    Intel(R) Math Kernel Library Version 11.0.5 Product Build 20130612 for Intel(R) 64 architecture applications
```

Linear Algebra PACKage Version 3.4.1

*Figure 1. Intel MKL version 11.0.5 is used within MATLAB R2014a*

**Step 2: Check if the Intel MKL version inside of MATLAB supports Intel Xeon Phi coprocessors**

Intel MKL has supported for Intel Xeon Phi coprocessor since release 11.0 for Linux OS, and since release 11.1 for Windows OS.

Some BLAS Level-3 and LAPACK functions are optimized to divide their computational work automatically between the host and the Intel Xeon Phi coprocessor. The automatic division of work is called automatic offload (AO). The list of AO enabled functions in Intel MKL is available from
Using the AO model, you can enable MATLAB working on Intel Xeon Phi coprocessor. Since the Intel MKL version in MATLAB 2014a for Windows OS is 11.0.5 and Intel Xeon Phi coprocessor Windows support requires Intel MKL 11.1 or later releases, in this example you must upgrade the Intel MKL version to 11.1 in MATLAB to enable Intel Xeon Phi coprocessor support.

If the version of Intel MKL inside of MATLAB is higher than 11.1, go to step 4.

**Step 3: Upgrade Intel MKL version in MATLAB**

If the versions is not higher than 11.1, you need to use mkl_rt.dll or create a custom dynamic library from the latest Intel MKL and supply it to MATLAB.

### 3.1 Use mkl_rt.dll

The Single Dynamic Library interface (SDL interface) mkl_rt library was introduced in Intel MKL 10.3. See the details in https://software.intel.com/en-us/articles/a-new-linking-model-single-dynamic-library-mkl_rt-since-intel-intel®-MKL-103

To replace the existing mkl.dll in MATLAB with a new mkl_rt.dll:

a. Open a “cmd” command line window and run the command:
   
   ```
   >>"C:\Program Files (x86)\Intel\Composer XE 2013 SP1\mkl\bin\mklvars.bat" intel64
   ```

b. Set the value of the variables BLAS_VERSION, LAPACK_VERSION to mkl_rt.dll
   
   ```
   >>set BLAS_VERSION=mkl_rt.dll
   >>set LAPACK_VERSION=mkl_rt.dll
   ```

c. Start MATLAB from the command window
   
   ```
   >>"C:\Program Files\MATLAB\R2014a\bin\matlab.exe"
   ```

Check the BLAS version information via “version -blas”. As shown in the figure below, you see the new version information.

![Figure 2. Use Intel MKL version 11.1.1 within MATLAB R2014a](https://software.intel.com/en-us/articles/intel-MKL-automatic-offload-enabled-functions-for-intel-xeon-phi-coprocessors)
3.2 Creation of custom dynamic library (Optional)


a. Download the matlab-custom-dll.zip (which only provides the BLAS functions listed in the example)

b. Copy the directory of `${MKL install directory}\tools\builder` to a working folder, for example, `C:\Users\yhu5\Desktop\Matlab\builder` and also extract the files from `matlab_custom_dll.zip` to the working folder

c. Build `mkl_custom.dll` from the command line via a makefile

Open a Windows command line window and set environment for using Microsoft Visual Studio 2010 x64 tools.
>`"C:\Program Files (x86)\Intel\Composer XE 2013 SP1\mkl\bin\mklvars.bat" intel64`
>`cd C:\Users\yhu5\Desktop\Matlab\builder`
>`nmake libintel64 export=blas_cdecl_list`

d. Set BLAS_VERSION to `mkl_custom.dll`
>`set BLAS_VERSION=C:\Users\yhu5\Desktop\Matlab\builder\mkl_custom.dll`
>`"C:\Program Files\MATLAB\R2014a\bin\matlab.exe"`

e. Check the BLAS version information using the “version –blas”, “version – modules” commands in MATLAB command window. You should see that the default blas library is invisible and the new `mkl_custom.dll` is loaded as in the figure below.
>`>> version -blas`
>`ans = Version information not found.`
>`>> version -modules`

![Figure 3. Use custom mkl dynamic library in MATLAB](image)

Step 4: Enable Intel MKL Automatic Offload (AO) in MATLAB

To enable MATLAB with Intel MKL AO are straightforward and general steps are as below:

a. Source environment using compilervars.sh or mklvars.sh intel64

b. Set MKL_MIC_MAX_MEMORY=16G; set MKL_MIC_ENABLE=1

c. Run matlab

In the example, you can use the same command window as in the previous step 3.1 and run the
below command:

>set Intel® MKL_MIC_MAX_MEMORY=16G
>set Intel® MKL_MIC_ENABLE=1
>"C:\Program Files\MATLAB\R2014a\bin\matlab.exe"

Figure 4. Set environment variable to enable MKL AO feature

Step 5: Verify the Intel MKL version and ensure that AO is enabled on the Intel Xeon Phi coprocessors

In opened MATLAB command window last step, run the version -blas and version –lapack, getenv('MKL_MIC_ENABLE') commands and check the output list to verify Intel MKL version.

The figure 5 shows that the Intel MKL version was upgraded to 11.1 update 1

Figure 5. The Intel MKL version in Matlab was upgraded to 11.1 update 1

Step 6: Compare performance

Furthermore, you can verify that AO was enabled and see the performance speedup for Intel Xeon Phi coprocessors.

6.1 Accelerate the common used matrix multiply A*B in MATLAB

since MATLAB use Intel MKL BLAS/LAPACK routines internally to perform the underlying computations, for example, matrix multiply A*B. you can run a simple test to see the performance change with or without MKL_MIC_ENABLE=1.

a. Save the below code into MATLAB m file-TestBlas.m

```matlab
A = rand(10000, 10000);
B = rand(10000, 10000);
```
tic
C = A*B;
toc

b. If you start a MATLAB session without setting MKL_MIC_ENABLE:
>"C:\Program Files (x86)\Intel\Composer XE 2013 SP1\mk\bin\mklvars.bat" intel64
>set BLAS_VERSION=mkl_rt.dll
>"C:\Program Files\MATLAB\R2014a\bin\matlab.exe"
In MATLAB Environment, open TestBLAS.m and click the ‘Run’ Button
MATLAB command window displays:
>> TestBlas
Elapsed time is 4.997849 seconds.
The CPU usage is shown below,

![Figure 6. Time measure and CPU usage without MKL_MIC_ENABLE](image)

Figure 6. Time measure and CPU usage without MKL_MIC_ENABLE

c. If you start a MATLAB session after setting MKL_MIC_ENABLE, the MATLAB command window displays:
>> TestBlas
Elapsed time is 1.869576 seconds

![Figure 7. Time measure and CPU usage, coprocessor cores usage with MKL_MIC_ENABLE](image)

Figure 7. Time measure and CPU usage, coprocessor cores usage with MKL_MIC_ENABLE
The performance was improved about 2.67 times. You can use ‘micscm’ to observe the Intel Xeon Phi core utilization. As figure 7 shows, both coprocessor cards work with the full cores.

6.2 Accelerate the BLAS function dgemm() in MATLAB (Optional)

MATLAB provide an example which call BLAS functions explicitly: dgemm() (C=A*B) via mex file at

This procedure adds performance measure information:

a. Create a source MEX-file containing the mexFunction gateway routine

The source code: matrixMultiplyM.c is attached.

Copy it to a working folder: for example c:/Users/yhu5/Desktop/Matlab, and change the Current Folder of MATLAB command window to point to it.

b. In MATLAB command window, select Intel C++ compiler, which support Intel Xeon Phi platform

>> mex -setup:'C:\Program Files\MATLAB\R2014a\bin\win64\mexopts\intel_c_13_vs2010.xml' C

c. In MATLAB command window, build a binary MEX-file using the mex command

>> mex -v -largeArrayDims matrixMultiplyM.c "C:\Program Files (x86)\Intel\Composer XE 2013 SP1\mkl\lib\intel64\mkl_rt.lib"

Figure 8. Build matrixMultiplyM binary

Once the matrixMultiplyM.mexw64 binary is built, you can keep it and use it like a common function. But please be sure MATLAB can search the directory of the binary when you call the function.

d. Test the matrixMultiplyM MEX-File with testBlas.m

A = rand(10000, 10000);  
B = rand(10000, 10000);  
tic  
X = matrixMultiplyM(A,B);
If you start a MATLAB without set MKL_MIC_ENABLE
>> set MKL_MIC_ENABLE="C:\Program Files (x86)\Intel\Composer XE 2013 SP1\Intel\MKL\bin\mklvars.bat"
intel64
>> set MKL_MIC_ENABLE="CMD:\Program Files (x86)\Intel\Composer XE 2013 SP1\Intel\MKL\bin\mklvars.bat"

Open TestBLAS.m and Click Run button
MATLAB Command Windows display:
>> TestBlas
Elapsed time is 4.958276 seconds.
CPU usage is shown below

Figure 9. Time measure and cpu usage without MKL_MIC_ENABLE setting

If you start MATLAB after setting MKL_MIC_ENABLE, click the “Run” button, the MATLAB Command Window displays:
>> TestBlas
Elapsed time is 1.837540 seconds.
You can also use ‘micscm’ to observe the Intel Xeon Phi core utilization. As figure 10 shows, both coprocessors are working with full cores.
The performance was improved about 2.69 times comparing to the computation without MKL_MIC_ENABLE setting.
Figure 10. Time measure, cpu usage and coprocessor cores usage with MKL_MIC_ENABLE=1

Note:
The description uses Windows* OS, but Linux* OS steps are similar. Consider using Intel64 version of library and MATLAB since Automatic Offload is supported only on the Intel® 64 architecture.

These instructions apply to all hardware configurations that meet the system requirements for both MATLAB R2014a and Intel MKL 11.1.1 and 11.1.2

The examples shown in the article were done using MATLAB 2014a, you expect them would also work for earlier MATLAB version, for example, MATLAB 2013a and MATLAB 2013b.

Summary
Intel MKL provides automatic offload (AO) feature for the Intel Xeon Phi coprocessor. With AO feature, certain MKL function can transfer part of the computation to Intel Xeon Phi coprocessor automatically. When problem sizes are large enough to amortize the cost of data transferring, the MKL functions performance can benefit from using both the host CPU and the Intel Xeon Phi coprocessor for computation. Because offloading happens transparently in AO, third-party software that uses Intel MKL functions can automatically benefit from this feature, easily making them to run faster on systems with Intel Xeon Phi coprocessor.

The article describes how to enable Intel MKL AO for MathWorks MATLAB on Intel Xeon Phi coprocessors system. The general steps are as below
1. Source environment using compilervars.sh or mklvars.sh intel64
2. Upgrade the Intel MKL version in MATLAB to the latest version supporting Intel Xeon Phi coprocessors
3. Set MKL_MIC_MAX_MEMORY=16G; set MKL_MIC_ENABLE=1
4. Run MATLAB

A simple test shows that on one test system with two Intel Xeon phi coprocessor, the common
used matrix multiplication within MATLAB (C = A*B) achieves a 2.6 times speedup when Intel MKL AO is enabled, comparing to doing the same computation on the cpu only.

Figure 11: The matrix multiply functions performance comparison w/o MKL_MIC_ENABLE support

Reference

Please send your question to intel.mkl@intel.com or Intel MKL Forum


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