White Paper

Title: Using Intel® MKL Automatic Offload on Intel® Xeon Phi™ Coprocessors

Overview

Intel® Math Kernel Library (MKL) includes a unique new Automatic Offload (AO) feature that enables computationally intensive Intel® MKL functions called in user code to benefit from attached Intel® Xeon Phi™ coprocessors automatically and transparently. This special feature allows users to leverage the additional computational resources provided by the coprocessor without changes to Intel MKL calls in their codes or modifications to the way they compile and link. AO allows programmers to use the same usage model they are familiar with, while enjoying performance scaling from multi-cores to many cores.

Because the coprocessor(s) are connected to the host system via Peripheral Component Interconnect Express (PCIe) , AO support is provided only for functions that involve sufficiently large problems and those having large computation versus data access ratios. For example, Intel MKL functions having computational complexities of greater than $O(n^3)$ versus $O(n^2)$ data accesses, as is the case with the many LAPACK functions and Level 3 BLAS, can amortize the time required to transfer the data with the significantly greater computation time , thus realizing an overall reduction in the time to solve the

problem which translates into a performance gain. Functions having these characteristics are good candidates for AO. In the current MKL release (11.0), the following Level-3 BLAS functions and LAPACK functions are AO-enabled:

- ?GEMM, ?SYMM, ?TRMM, and ?TRSM
- LU, QR, Cholesky factorizations

Enabling and Disabling Automatic Offload

Automatic Offload can be enabled using a single call of a support function, or setting an environment variable. Compiler pragmas are not needed, and users can compile and link the code the usual way.

The following text shows how to enable AO in FORTRAN or C code,

```
rc = mkl_mic_enable( )
```

Alternatively, it can be enabled using an environment variable.

```
MKL_MIC_ENABLE=1
```

Once enabled, Intel MKL may automatically offload computation to one or more Intel Xeon Phi coprocessors.

The offloading of computations is transparent to the user in the sense that the Intel MKL runtime decides how much work to offload. Depending on the problem size and the current status of coprocessor(s), the library may choose to run all the computational work, part of it, or none of it on coprocessor(s). Likewise, the library may decide to use all of the coprocessors available or to use only one coprocessor. For users who would like to control how much of a problem is offloaded, Intel MKL provides mechanisms to fine-tune how the work should be divided between the host and coprocessor(s). See the "How to Control Work Division" section below for the information.

Offloading is transparent also in the sense that if no coprocessor is present, the same executable still works. It simply runs all the computation on the host, as usual, without any penalty.

To disable Automatic Offload, simply insert a call to the support function `mkl_mic_disable()`. It can be re-enabled by calling `mkl_mic_enable()`.

Controlling Work Division

For the supported Level 3 BLAS functions (?GEMM, ?TRMM, ?TRSM), Intel MKL provides support functions and environment variables to allow users to override the default work division decided by the Intel MKL runtime, as well as to query the current work division settings and the number of available coprocessors. Refer to the reference manual for detailed documentation for each of supported functions. Note that these work division controls work differently for LAPACK functions. For LAPACK functions, any non-zero value of work division is interpreted as 100%.

The table below gives a few examples showing how to set and manage the division of work between the host and coprocessor(s).

Examples	Notes
mkl_mic_set_workdivision (MKL_TARGET_MIC, -1, MKL_MIC_AUTO_WORKDIVISION)	Let the runtime to decide how much work to offload to all cards. The runtime system decides which card(s) to use.
mkl_mic_set_workdivision (MKL_TARGET_MIC, -1, 0.5)	Offload 50% of computation to all cards. The runtime system decides which cards to use.
mkl_mic_set_workdivision (MKL_TARGET_MIC, 0, 0.5)	Offload 50% of computation only to the 1st card.
mkl_mic_set_workdivision (MKL_TARGET_HOST, 0, 0.5)	Keep 50% of computation on the host; offload the rest to all cards. (In this case the second argument is ignored)
mkl_mic_get_workdivision (MKL_TARGET_MIC, 0, &wd)	Query how much work was specified for the 1st card.
mkl_mic_get_device_count ( )	Find out how many cards available on the system.

Work division can also be controlled using environment variables. See the examples in the table below. Note that support functions always take precedence over environment variables.

Examples	Notes
MKL_MIC_WORKDIVISION=MIC_AUTO_WORKDIVISION	Let the runtime to decide how much work to offload to all cards. The runtime system decides which card(s) to use.
MKL_MIC_WORKDIVISION=0.5	Offload 50% of computation to all cards. The runtime system decides which cards to use.
MKL_MIC0_WORKDIVISION=0.5	Offload 50% of computation only to the 1st card.
MKL_HOST_WORKDIVISION=0.5	Keep 50% of computation on the host; offload the rest to all cards.

Tips on Using Automatic Offload

These coding tips help users to better understand AO and to get the most out of it.

Automatic Offload works better when matrix size is right

Matrix size is critical for AO to get good performance. In fact, AO does not even start when matrices are too small (row or column size is smaller than 2048[1]). This is because in this situation the overhead of data transferring overshadows any performance benefit offloading can bring. If matrices are sufficiently large, then the best performance is typically achieved with square matrices.

Reserve maximum amount of memory to be used by Automatic Offload on the coprocessor

Intel MKL allocates additional memory for each process that performs Automatic Offload computations. To reduce the overhead of buffer initialization and data transferring, users can limit coprocessor memory used by Automatic Offload. This can be done by either calling the support function `mkl_mic_set_max_memory()`, or by setting the environment variable `MKL_MIC_MAX_MEMORY`. For example, the following set the limit for the first coprocessor to 4 GB.

```
rc = mkl_mic_set_max_memory(MKL_TARGET_MIC, 0, 4G)
```

A call to the function `mkl_mic_free_memory()` can be used to release the reserved memory on a specific coprocessor.

[1] The threshold for AO may be different in future releases.

Use the run time offload report to understand Automatic Offloading

Because Automatic Offload is transparent, by default users do not know whether it is happening and how it is happening. But sometimes (e.g. for debugging) it helps to know what is going on behind the scene. For this purpose, the runtime of Intel MKL can generate an offload report for Automatic Offload, while the program is running. To view the profiling report, the environment variable `OFFLOAD_REPORT` has to be set to 1 or 2 before the execution. The value controls the level of information included in the report, with the value 2 gives more detailed information.

When `OFFLOAD_REPORT` is set, users can use the function `mkl_mic_set_offload_report()` to dynamically turn on/off the report. This function takes a single integer value. A non-zero value turns on the report. And then another call with a zero value turns off the report.

Use Automatic Offload and Compiler Assisted Offload in the same program

In addition to Automatic Offload, Intel MKL supports Compiler Assisted Offload (CAO). That is, offload can be explicitly specified using compiler pragmas provided in Intel® Fortran Compiler XE and Intel® C/C++ Compiler XE. CAO requires more effort from programmers, but it provides more control and is more flexible. In sophisticated applications, cases may arise where mixing AO and CAO in one application is necessary. Intel MKL does support this usage model. However, when AO and CAO are used in the same program, users need to explicitly specify work division for AO-aware functions using support functions or environment variables. Otherwise, the default work division setting is to run all the work on the host and not to offload.

Force execution to fail if offload fails

Intel MKL AO will automatically do all the computations on the host if the offload runtime cannot find a coprocessor or cannot initialize it properly. This means that users cannot be sure whether a computation is done on a coprocessor or on the host. Users can override this default behavior by setting the environment variable `MKL_MIC_DISABLE_HOST_FALLBACK`.

```
MKL_MIC_DISABLE_HOST_FALLBACK=1
```

Setting `MKL_MIC_DISABLE_HOST_FALLBACK` will cause the program to exit with the error message, "Could not enable Automatic Offload", if offload is attempted but fails.

Threading control in Automatic Offload

Threading on the host and on coprocessors can be controlled separately using two sets of OpenMP environment variables. Since AO may spawn computations on both on the host and coprocessors, it is usually necessary to set these environment variables on both sides for optimal performance.

OpenMP threading controls provide users a flexible way to set the maximum number of threads allowed for use during program execution and also to setting thread affinity. The environment variables are:

Host	Coprocessor
OMP_NUM_THREADS	MIC_OMP_NUM_THREADS
KMP_AFFINITY	MIC_KMP_AFFINITY

The environment variables for the coprocessor are usually prefixed with "MIC_". But this prefix is in fact customizable and users can choose to use a different prefix by setting another environment variable MIC_ENV_PREFIX. These environment variables are set by the user on the host side. In the case of AO, the Intel MKL runtime interprets them and automatically passes them to the environment on the coprocessor.

On a coprocessor, one core is always being used for the micro operating system (OS). Tasks such as data transfer and housekeeping also run on this core. For performance reasons, offloaded execution should avoid using the core running the micro OS. The recommendation is to set thread affinity explicitly on the coprocessor (via MIC_KMP_AFFINITY) to use all available cores except the one running the OS. For the portion of the work run on the host, it is recommended that KMP_AFFINITY be set to prevent thread migration between cores and to avoid the loss of cache locality.

To better clarify some of the concepts described, let's consider an example. The following environment variable settings example shows the appropriate thread control settings for a system with a 60-core coprocessor (4 threads per core) and an 8-core host (1 thread per core):

```
OMP_NUM_THREADS=8
KMP_AFFINITY=granularity=fine,compact,1,0
MIC_OMP_NUM_THREAD=236
MIC_KMP_AFFINITY=explicit,granularity=fine,proclist=[1-236:1]
MIC_ENV_PREFIX=MIC_
```

Conclusion

Automatic Offload is the simplest way to use Intel MKL on systems with Intel Xeon Phi coprocessors. It enables users to take advantage of both host and coprocessor(s) without any code changes. If coprocessor(s) exist, Intel MKL makes an intelligent decision on what operations are worth offloading, and the appropriate work divisions between the host and the coprocessor(s). The library also

transparently handles data transfer and manages remote execution. If no coprocessor present, execution falls back to the host without any penalty.

Intel MKL provides service functions and environment variables to control work divisions between the host and the coprocessors for users wanting a finer level of Automatic Offload controls.

It is also possible to use Automatic Offload and Compiler Assisted Offload for Intel MKL functions in the same application.

Intel MKL is committed to bringing breakthrough performance for highly parallel applications on Intel Xeon Phi products. The Automatic Offload feature makes this happen smoothly by ensuring a consistent programming model on Intel Xeon and Intel Xeon Phi. Intel MKL users can continue to use the same code and their familiar tools and libraries while enjoying the performance scaling from multi-cores to many cores.

Additional Resources

The following documents provide the complete documentation on support functions and environment variables discussed in this article.

- Intel Math Kernel Library Reference Manual
- Intel Math Kernel Library User's Guide

Acknowledgements

This article is based on a technical presentation by Intel MKL architect and principal engineer Greg Henry (greg.henry@intel.com).

Thanks to Shane Story (shane.story@intel.com), Todd Rosenquist (todd.rosenquist@intel.com), Ying Song (ying.song@intel.com), Larry McGlinchy (larry.j.mcglinchy@intel.com), and Roger Herrick (roger.i.herrick.jr@intel.com) for reviewing this document.

About the Author

Zhang Zhang (zhang.zhang@intel.com) is a Technical Consulting Engineer supporting Intel software development tools, with a focus on enabling ISVs to use Intel Math Kernel Library and other software products on the Intel architectures.

Notices

DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

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

Optimization Notice

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

NOTE: additional notices and disclaimers may be needed, depending on the content of the collateral. Generally, they can be found in the following areas, and must be appended to the Notices section of the paper.

General Notices: *Notices placed in all materials distributed or released by Intel*

Benchmarking and Performance Disclaimers: Disclaimers for Intel materials that use benchmarks or make performance claims.

Technical Collateral Disclaimers: *Disclaimers that should be included in Intel technical materials that describe the form, fit or function of Intel products.*

Technology Notices: *Notices for Intel materials when the benefits or features of a technology or program are described. Note for technology disclaimers - if every product being discussed (e.g., ACER ULV) has the particular technology/feature, then you can remove the requirements statement in the disclaimer. If you have multiple technical disclaimers, you can consolidate the "your performance may vary" statements and only put in a single "your mileage may vary".*