Intel MIC Programming Workshop: MKL
Dr. Momme Allalen (LRZ)
June, 26-28, 2017 @ LRZ
Agenda

- A quick overview of Intel MKL
- Usage of MKL in Accelerator mode (KNC)
  - Compiler Assisted Offload
  - Automatic Offload
  - Native Execution
- Usage of MKL on KNL
- Hands-on & Performance
- Useful links where do we find more information?
What is the Intel MKL?

- Math library for C and Fortran, Includes:
  - BLAS, BLAS95 and Square BLAS
  - LAPACK
  - ScaLAPACK with BLACS
  - FFT and FFTW
  - Vector Math, Vector Statistics Functions
  - …

- Containing optimised routines
  - For Intel CPUs and MIC architecture

- All MKL functions are supported on Xeon Phi
  But optimised at different levels
Execution Models on Intel Xeon Phi Architectures

Multicore Xeon Or KNL

MKL AO & CAO

Symmetric Codes with balanced needs

Many Core Hosted (KNC) highly-parallel codes

Offload Codes with highly-parallel phases

KNL or Multicore Hosted General purpose serial and parallel computing

MKL Native
MKL Usage In Accelerator Mode (KNC)

• Compiler Assisted Offload
  • Offloading is explicitly controlled by compiler pragmas or directives.
  • All MKL functions can be inserted inside offload region to run on the Xeon Phi (In comparison, only a subset of MKL is subject to AO).
  • More flexibility in data transfer and remote execution management.

• Automatic Offload Mode
  • MKL functions are automatically offloaded to the accelerator.
  • MKL decides:
    • When to offload
    • Work division between host and targets
  • Data is managed automatically

• Native Execution
MKL Usage In Accelerator Mode (KNC)

- Compiler Assisted Offload
  - Offloading is explicitly controlled by compiler pragmas or directives.
  - All MKL functions can be inserted inside offload region to run on the Xeon Phi (In comparison, only a subset of MKL is subject to AO).
  - More flexibility in data transfer and remote execution management.

- Automatic Offload Mode
  - MKL functions are automatically offloaded to the accelerator.
  - MKL decides:
    - When to offload
    - Work division between host and targets
  - Data is managed automatically

- Native Execution
How to use CAO

- The same way you would offload any function call to MIC
- An example in C:

```c
#pragma offload target(mic) \ 
   in(transa, transb, N, alpha, beta) \ 
   in(A:length(matrix_elements)) \ 
   in(B:length(matrix_elements)) \ 
   in(C:length(matrix_elements)) \ 
   out(C:length(matrix_elements) alloc_if(0))
{
   sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N,&beta, C, &N);
}
```
An example in Fortran:

!DEC$ ATTRIBUTES OFFLOAD : TARGET( MIC ) :: SGEMM
!DEC$ OMP OFFLOAD TARGET( MIC ) &
!DEC$ IN( TRANSA, TRANSB, M, N, K, ALPHA, BETA, LDA, LDB, LDC ), &
!DEC$ IN( A: LENGTH( NCOLA * LDA ) ), &
!DEC$ IN( B: LENGTH( NCOLB * LDB ) ), &
!DEC$ INOUT( C: LENGTH( N * LDC ) )

!$OMP PARALLEL SECTIONS
!$OMP SECTION
   CALL SGEMM( TRANSA, TRANSB, M, N, K, ALPHA, &A, LDA, B, LDB BETA, C, LDC )
!$OMP END PARALLEL SECTIONS
Tips for Using Compiler Automatic Offload

- Use larger (>2MB) pages for data transferring;

  e.g: ~$export MIC_USE_2MB_BUFFERS=50M

- This means that for any array allocation larger than 50MB, uses huge pages
MKL Usage In Accelerator Mode (KNC)

- Compiler Assisted Offload
  - Offloading is explicitly controlled by compiler pragmas or directives.
  - All MKL functions can be inserted inside offload region to run on the Xeon Phi (In comparison, only a subset of MKL is subject to AO).
  - More flexibility in data transfer and remote execution management.

- Automatic Offload Mode
  - MKL functions are automatically offloaded to the accelerator.
  - MKL decides:
    - When to offload
    - Work division between host and targets
  - Data is managed automatically

- Native Execution
How to Use Automatic Offload

- User does not have to change the code at all
- Either by calling the function mkl_mic_enable() or by setting the following environment variable

  ```
  ~$export MKL_MIC_ENABLE=1
  ```

- In Intel MKL 11.0.2 the following functions are enabled for automatic offload:

  **Level-3 BLAS functions**
  - *GEMM* (for $M,N > 2048$, $k > 256$)
  - *TRSM* (for $M,N > 3072$)
  - *TRMM* (for $M,N > 3072$)
  - *SYMM* (for $M,N > 2048$)

  **LAPACK functions**
  - LU ($M,N > 8192$)
  - QR
  - Cholesky

allalen@lrz.de
Intel MIC Programming Workshop @ LRZ
How to Use Automatic Offload

• **Blas only**: work can be divided between host and device using

  \[
  \text{mkl\_mic\_set\_workdivision(TARGET\_TYPE, TARGET\_NUMBER, WORK\_RATIO)}
  \]

• What if there doesn’t exist a MIC card in the system?
  • Runs on the host as usual **without any penalty** !!

• Users can use AO for some MKL calls and use CAO for others in the same program
  • Only supported by Intel compilers
  • Work division must be set explicitly for AO, otherwise, all MKL AO calls are executed on the host
#include “mkl.h”
err = mkl_mic_enable();

//Offload all work on the Xeon Phi
err = mkl_mic_set_workdivision(MKL_TARGET_HOST, MIC_HOST_DEVICE, 0, 0);

//Let MKL decide of the amount of work to offload on coprocessor 0
err = mkl_mic_set_workdivision(MKL_TARGET_MIC, 0, MIC_AUTO_WORKDIVISION);

//Offload 50% of work on coprocessor 0
err = mkl_mic_set_workdivision(MKL_TARGET_MIC, 0, 0.5);

//Get amount of work on coprocessor 0
err = mkl_mic_get_workdivision(MKL_TARGET_MIC, 0, &wd);
Tips for Using Automatic Offload

- AO works only when matrix sizes are right
  - SGEMM: Offloading only when M, N > 2048
  - Square matrices give much better performance

- These settings may produce better results for SGEMM calculation for 60-core coprocessor:
  
  ```
  export MIC_USE_2MB_BUFFERS=16K
  export MIC_OMP_NUM_THREADS=240
  export MIC_ENV_PREFIX=MIC
  export MIC_KMP_AFFINITY=compact,granularity=fine
  export MIC_PLACE_THREADS=60C,4t
  ```

- Work division settings are just hints to MKL runtime

  Threading control tips:
  
  Prevent thread migration on the host using:
  
  ```
  export KMP_AFFINITY=granularity=fine, compact, 1,0
  ```
MKL Usage In Accelerator Mode (KNC)

• Compiler Assisted Offload
  • Offloading is explicitly controlled by compiler pragmas or directives.
  • All MKL functions can be inserted inside offload region to run on the Xeon Phi (In comparison, only a subset of MKL is subject to AO).
  • More flexibility in data transfer and remote execution management.

• Automatic Offload Mode
  • MKL functions are automatically offloaded to the accelerator.
  • MKL decides:
    • When to offload
    • Work division between host and targets
  • Data is managed automatically

• Native Execution
Native Execution

- In order to use Intel MKL in a native application, an additional argument `-mkl` is required with the compiler option `-mmic`.

- Native applications with Intel MKL functions operate just like native applications with user-defined functions.

```
$ icc -O3 -mmic -mkl sgemm.c -o sgemm.exe
```
Compile to use the Intel MKL

- Compile using \texttt{–mkl} flag
  - \texttt{-mkl=parallel} (default) for parallel execution
  - \texttt{-mkl=sequential} for sequential execution

- AO: The same way of building code on Xeon:
  - user@host $ icc -O3 -mkl sgemm.c -o sgemm.exe

- Native using \texttt{-mmic}
  - user@host $ ifort –mmic –mkl myProgram.c –o myExec.mic

- MKL can also be used in native mode if compiled with \texttt{-mmic}
Involving different MKL versions

<table>
<thead>
<tr>
<th>MKL Version</th>
<th>Link flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single thread, sequential</td>
<td>-mkl=sequential</td>
</tr>
<tr>
<td>Single thread MPI, Sequential</td>
<td>-mkl=cluster</td>
</tr>
<tr>
<td>Multi thread</td>
<td>-mkl=parallel or -mkl</td>
</tr>
</tbody>
</table>
More code examples:

- module show mkl
- $MKLROOT/examples/examples_examples_mic.tgz
  - sgemm SGEMM example
  - sgemm_f SGEMM example (Fortran 90)
  - fft complex-to-complex 1D FFT
  - solverc Pardiso examples
  - sgaussian single precision Gaussian RNG
  - dgaussian double precision Gaussian RNG
  - ...
Which Model to Choose:

- **native execution for**
  - Highly parallel code
  - Using MIC as independency compute nodes

- **AO if**
  - Sufficient Byte/FLOP ratio makes offload beneficial
  - Using Level-3 BLAS functions: GEMM, TRMM, TRSM

- **CAO if**
  - There is enough computations to offset data transfer overhead
  - Transferred data can be reused by multiple operations

Usage of MKL on KNL

- up to version 2017 MKL automatically try to use MCDRAM
  - with unlimited access to MCDRAM

- In order to restrict how much MCDRAM MKL uses is possible
  - Environment variable (in MB)
    MKL_FAST_MEMORY_LIMIT=40
  - Function call (in MB):
    mkl_set_memory_limit(MKL_MEM_MCDRAM, 40)
• Use the “intel64” MKL version
• “mic” version is for KNC only
• Usage:
  • $host: icpc -O3 -xMIC-AVX512 -lmemkind -qopenmp -mkl -qopt-report=0 ...
# Intel® MKL 2018 Beta (18 April 2017)

| DPD200576142 | Improved the performance of functions DORCSD2BY1 routine |
| DPD200418144 | Fixed the wrong result issue when calling zgetrf by MKL_DIRECT_CALL |
| DPD200592232 | Fixed the scaling improvement of 3D r2c and c2r Cluster FFT, for some specific problems size |
| DPD200592231 | Improved general performance on Intel® Xeon Phi™ product family x200 |
| DPD200592229 | MKL FFT: When 3D FFT scale is not 1, performance drops on KNL but not HSX for the c2r backward transform |
| DPD200591316 | Fixed the problem with MKL Libraries mkl_* . Dll have to load their dependence on the absolute path of the folder where the DLL itself is located |
| DPD200592227 | Fixed the problem when MKL FFT selects 1 thread per core on KNL regardless how many threads are requested by environment |
Lab: MKL
Memory Allocation: Data Alignment

• Compiler-assisted offload
  • Memory alignment is *inherited from host!*

• General memory alignment (SIMD vectorisation)
  • Align buffers (leading dimension) to a multiple of vector width (64 Byte)
    • `mk1_malloc`, `_mm_malloc` (`_aligned_malloc`),
      `tbb::scalable_aligned_malloc`, ...

June, 26-28, 2017

Intel MIC Programming Workshop @ LRZ

callalen@lrz.de
void * darray;
int workspace;
int alignment = 64;
...
darray = mkl_malloc(sizeof(double) * workspace, alignment);
...
mkl_free(darray);
Performance of many Intel MKL routines improves when input and output data reside in memory allocated with 2 MB pages

—> Address more memory with less pages, reduce overhead of translating between host- and MIC address spaces

# Allocate all pointer-based variables with run-time
# length > 64 KB in 2 MB pages:
$ export MIC_USE_2MB_BUFFERS=64K
Native:
KMP_AFFINITY=balanced
OMP_NUM_THREADS=244

Compiler-Assisted Offload:
MIC_ENV_PREFIX=MIC
MIC_KMP_AFFINITY=balanced
MIC_OMP_NUM_THREADS=240
MIC_USE_2MB_BUFFERS=64K

Automatic Offload:
MKL_MIC_ENABLE=1
OFFLOAD_DEVICES=<list>
MKL_MIC_MAX_MEMORY=2GB
MIC_ENV_PREFIX=MIC
MIC_OMP_NUM_THREADS=240
MIC_KMP_AFFINITY=balanced
+ Compiler-Assisted Offload:
OFFLOAD_ENABLE_ORSL=1

**Memory Allocation: Page Size**

**KMP_AFFINITY**=
- Host: e.g., compact,1
- Coprocessor: balanced

**MIC_ENV_PREFIX=MIC; MIC_KMP_AFFINITY**=
- Coprocessor (CAO): balanced

**KMP_PLACE_THREADS**
- Note: does not replace **KMP_AFFINITY**
- Helps to set/achieve pinning on e.g., 60 cores with 3 threads each

**kmp_* (or mk1_*)** functions take precedence over corresponding env. variables
More MKL documentation

- Performance charts online: http://software.intel.com/en-us/intel-mkl#pid12768-1295
- https://software.intel.com/en-us/node/528430
- https://www.nersc.gov/assets/MKL_for_MIC.pdf
Thank you.