Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Intel MKL @ MIC Architecture

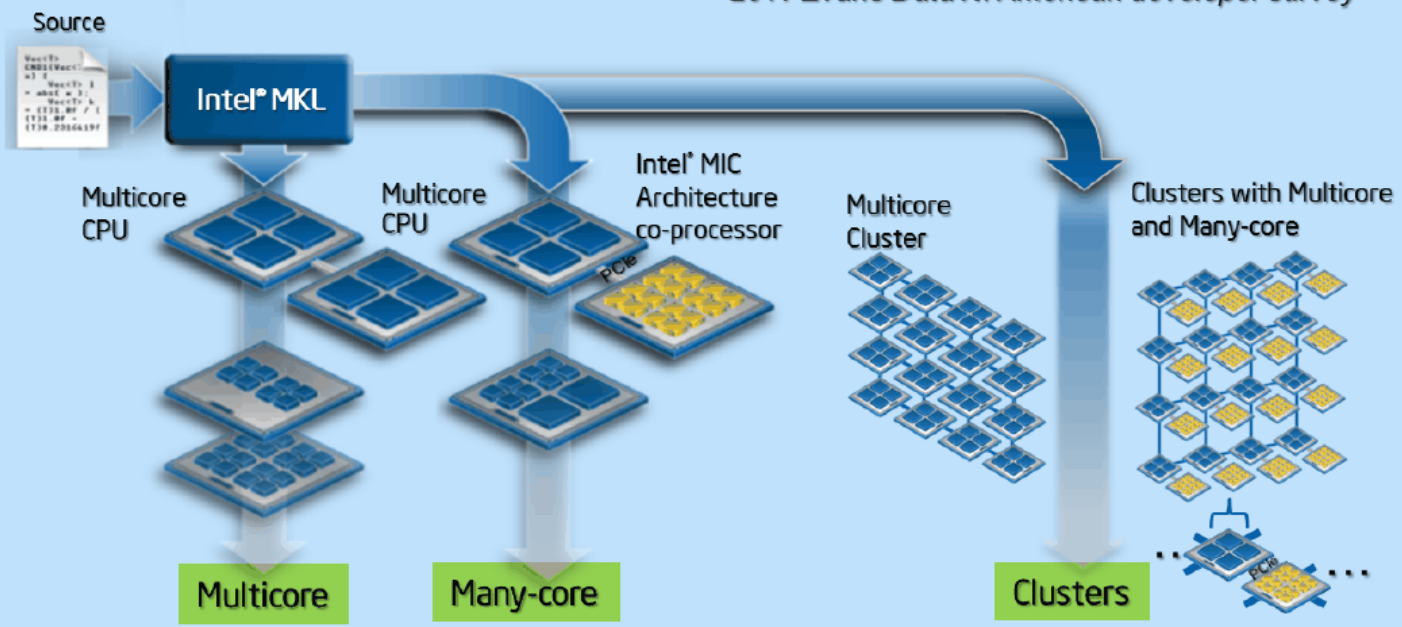
Dr. Momme Allalen, LRZ, allalen@lrz.de
PRACE PATC: Intel MIC&GPU Programming Workshop

What Is Intel Math Kernel Library?

Intel® MKL is industry's leading math library *

Linear Algebra	Fast Fourier Transforms	Vector Math	Vector Random Number Generators	Summary Statistics	Data Fitting
•BLAS •LAPACK •Sparse solvers •ScaLAPACK	•Multidimensional (up to 7D) •FFTW interfaces •Cluster FFT	•Trigonometric •Hyperbolic •Exponential, Logarithmic •Power / Root •Rounding	•Congruential •Recursive •Wichmann-Hill •Mersenne Twister •Sobol •Neiderreiter •Non-deterministic	•Kurtosis •Variation coefficient •Quantiles, order statistics •Min/max •Variance-covariance •...	•Splines •Interpolation •Cell search

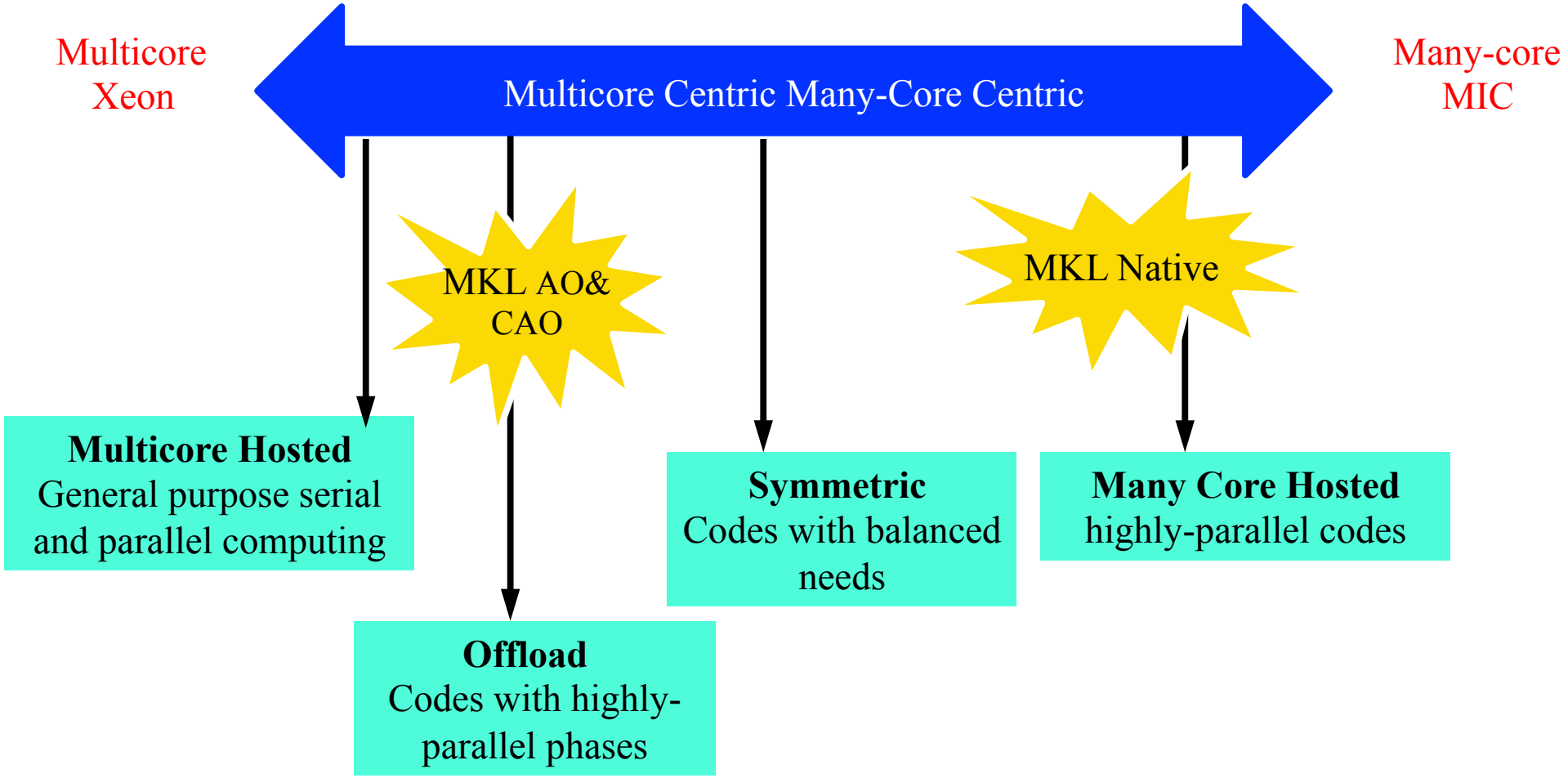
** 2011 Evans Data N. American developer survey*



What is the Intel MKL ?

- Math library for C and Fortran
- Includes
  - BLAS
  - LAPACK
  - ScaLAPACK
  - FFTW
  - …
- Containing optimized routines
  - For Intel CPUs and MIC architecture

Execution Models on Intel MIC Architectures



MKL Usage In Accelerator Mode

- 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 decision 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:

```
#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);
 }
```

How to use CAO

- 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;

  MIC_USE_2MB_BUFFERS

- Enable huge paging for memory allocation on the target

  libhugetlbfs.so (helps to get good performance!)

How to Use Automatic Offload

- Using Automatic Offload is easy
- Either by calling the function mkl_mic_enable() or by setting the following environment variable

```
$export MKL_MIC_ENABLE=1
```

- Work can be divided between host and device using

```
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** !!

Automatic Offload Mode Example

```
#include “mkl.h”
err = mkl_mic_enable();

//Offload all work on the Xeon Phi
err = mkl_mic_set_workdivsion (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_workdevision(MKL_TARGET_MIC, 0, 0.5);

//Get amount of work on coprocessor 0
err = mkl_mic_get_workdevision(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 61-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: export KMP_AFFINITY=granularity=fine, compact, 1,0

Using AO and CAO in the Same Program

- 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

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.

Compile to use the Intel MKL

- Compile using –mkl flag
  - -mkl=parallel(default) for parallel execution
  - -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 -mmic
  - user@host $ ifort –mmic –mkl myProgram.c –o myExec.mic

- NB:
  - MKL can also be used in native mode if compiled with -mmic

Using AO and CAO in the Same Program

- 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

MKL hands-on

More code examples:

- module show mkl
- $MKL_BASE/examples/mic_ao
- $MKL_base/examples/mic_offload
  - 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 coprocessors 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

More MKL documentation

- Tips, app notes, etc. can also be found on the Intel Many Integrated Core Community website:

  https://mic-dev.intel.com/

- MKL documentation at LRZ:

  $ module show mkl

  $ echo $MKL_DOC

  $ ls $MKL_DOC/mkl_userguide