Intel® Many Integrated Core (Intel MIC) Architecture

Targeted at highly parallel HPC workloads
- Physics, Chemistry, Biology, Financial Services

Power efficient cores, support for parallelism
- Cores: less speculation, threads, wider SIMD
- Scalability: high BW on die interconnect and memory

General Purpose Programming Environment
- Runs Linux (full service, open source OS)
- Runs applications written in Fortran, C, C++, ...
- Supports X86 memory model, IEEE 754
- x86 collateral (libraries, compilers, Intel® VTune™ debuggers, etc)
Knights Corner Core

X86 specific logic < 2% of core + L2 area
Summary

Intel® Xeon Phi™ coprocessor provides:

- Performance and Performance/Watt for highly parallel HPC with cores, threads, wide-SIMD, caches, memory BW

Intel Architecture
- general purpose programming environment
- advanced power management technology

KNC delivers programmability and performance/watt for highly parallel HPC
Intel® MPI Library Overview

- Intel is a leading vendor of MPI implementations and tools
- Optimized MPI application performance
  - Application-specific tuning
  - Automatic tuning
- Lower latency
  - Industry leading latency
- Interconnect Independence & Runtime Selection
  - Multi-vendor interoperability
  - Performance optimized support for the latest OFED capabilities through DAPL 2.0
- More robust MPI applications
  - Seamless interoperability with Intel® Trace Analyzer and Collector
Preserve Your Development Investment
Common Tools and Programming Models for Parallelism

Develop Using Parallel Models that Support Heterogeneous Computing
Spectrum of Programming Models and Mindsets

Multi-Core Centric

Multi-Core Hosted
General purpose serial and parallel computing

Symmetric
Codes with balanced needs

Many Core Hosted
Highly-parallel codes

Offload
Codes with highly-parallel phases

Main()
Foo()
MPI_*()

Main()
Foo()
MPI_*()

Main()
Foo()
MPI_*()

Main()
Foo()
MPI_*()

Main()
Foo()
MPI_*()

Range of models to meet application needs

Multi-core (Xeon)

Intel® Xeon Phi™

Intel® Xeon Phi™ Architecture
Software & Services Group, Developer Relations Division

Copyright© 2011, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Coprocessor only Programming Model

- MPI ranks on Intel® Xeon Phi™ (only)
- All messages into/out of coprocessors
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads used directly within MPI processes

Build Intel® Xeon Phi™ binary using Intel® compiler.

Upload the binary to the Intel® Xeon Phi™.

Run instances of the MPI application on Intel® Xeon Phi™ nodes.
Symmetric Programming Model

- MPI ranks on Intel® Xeon Phi™ Architecture and host CPUs
- Messages to/from any core
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* used directly within MPI processes

Build Intel® 64 and Intel® Xeon Phi™ binaries by using the resp. compilers targeting Intel® 64 and Intel® Xeon Phi™.

Upload the Intel® Xeon Phi™ binary to the Intel® Xeon Phi™ Architecture.

Run instances of the MPI application on different mixed nodes.
MPI+Offload Programming Model

- MPI ranks on Intel® Xeon® processors (only)
- All messages into/out of host CPUs
- Offload models used to accelerate MPI ranks
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* within Intel® Xeon Phi™

Build Intel® 64 executable with included offload by using the Intel® 64 compiler.

Run instances of the MPI application on the host, offloading code onto coprocessor.

Advantages of more cores and wider SIMD for certain applications
MKL on Intel® Xeon Phi™

- Intel® MKL usage models on Intel® Xeon Phi™
  - Automatic Offload
  - Compiler Assisted Offload
  - Native Execution
Intel® MKL is industry’s leading math library *

Linear Algebra
- BLAS
- LAPACK
- Sparse solvers
- ScalAPACK

Fast Fourier Transforms
- Multidimensional (up to 7D)
- FFTW interfaces
- Cluster FFT

Vector Math
- Trigonometric
- Hyperbolic
- Exponential, Logarithmic
- Power / Root
- Rounding

Vector Random Number Generators
- Congruential
- Recursive
- Wichmann-Hill
- Mersenne Twister
- Sobol
- Neiderreiter
- Non-deterministic

Summary Statistics
- Kurtosis
- Variation coefficient
- Quantiles, order statistics
- Min/Max
- Variance-covariance
- ...

Data Fitting
- Splines
- Interpolation
- Cell search

* 2011 & 2012 Evans Data N. American developer surveys
Intel® MKL Supports for Intel® Xeon Phi™ Coprocessor

• Intel® MKL 11.0 supports the Intel® Xeon Phi™ coprocessor.

• Heterogeneous computing
  o Takes advantage of both multicore host and many-core coprocessors

• Optimized for wider (512-bit) SIMD instructions

• Flexible usage models:
  o Automatic Offload: Offers transparent heterogeneous computing

Using Intel® MKL on Intel® Xeon Phi™
• Performance scales from multicore to many-cores
• Familiarity of architecture and programming models
• Code re-use, Faster time-to-market
Usage Models on Intel® Xeon Phi™ Coprocessor

• Automatic Offload
  o No code changes required
  o Automatically uses both host and target
  o Transparent data transfer and execution management

• Compiler Assisted Offload
  o Explicit controls of data transfer and remote execution using compiler offload pragmas/directives
  o Can be used together with Automatic Offload

• Native Execution
  o Uses the coprocessors as independent nodes
  o Input data is copied to targets in advance
Execution Models

Multicore Hosted

General purpose serial and parallel computing

Symmetric

Codes with balanced needs

Offload

Codes with highly-parallel phases

Multicore Centric Many-Core Centric

MKL AO & CAO

Many-core Hosted

Highly-parallel codes

Many Core Hosted

Many-core (Intel® Xeon Phi™)

Multicore Hosted

General purpose serial and parallel computing

Symmetric

Codes with balanced needs

Offload

Codes with highly-parallel phases

Many-Core Centric

MKL AO & CAO

Multicore (Intel® Xeon®)
Automatic Offload (AO)

• Offloading is automatic and transparent
• By default, Intel MKL decides:
  o When to offload
  o Work division between host and targets
• Users enjoy host and target parallelism automatically
• Users can still control work division to fine tune performance
How to Use Automatic Offload

• Using Automatic Offload is easy

```plaintext
Call a function: mkl_mic_enable()
```

or

```plaintext
Set an env variable:
MKL_MIC_ENABLE=1
```

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

• Can be temporarily disabled via mkl_mic_disable()
Automatic Offload Enabled Functions

• A selective set of MKL functions are AO enabled.
  o Only functions with sufficient computation to offset data transfer overhead are subject to AO

• In 11.0.1, AO enabled functions include:
  o Level-3 BLAS: ?GEMM, ?TRSM, ?TRMM
  o LAPACK 3 Amigos: ?GETRF, ?POTRF, ?GEQRF
  o AO support will be expanded in future updates.
Compiler Assisted Offload (CAO)

- Offloading is explicitly controlled by compiler pragmas or directives.

- All MKL functions can be offloaded in CAO.
  - In comparison, only a subset of MKL is subject to AO.

- Can leverage the full potential of compiler’s offloading facility.

- More flexibility in data transfer and remote execution management.
  - A big advantage is data persistence: Reusing transferred data for multiple operations.
How to Use Compiler Assisted Offload

- The same way you would offload any function call to the coprocessor.
- 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)) \ 
  inout(C:length(matrix_elements))
{ sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, 
  &beta, C, &N); }
```
How to Use Compiler Assisted Offload

• An example in Fortran:

!DEC$ ATTRIBUTES OFFLOAD : MIC :: SGEMM
!DEC$ 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 ) ), INOUT( C: LENGTH( N * LDC ) )
CALL SGEMM( TRANSA, TRANSB, M, N, K, ALPHA, &
A, LDA, B, LDB BETA, C, LDC )
Native Execution

• Programs can be built to run only on the coprocessor by using the `-mmic` build option.

• Tips of using MKL in native execution:
  - Use all threads to get best performance (for 60-core coprocessor)
    
    ```
    MIC_OMP_NUM_THREADS=240
    ```
  - Thread affinity setting
    
    ```
    KMP_AFFINITY=explicit,proclist=[1-240:1,0,241,242,243],granularity=fine
    ```
  - Use huge pages for memory allocation (More details on the “Intel Developer Zone”):
    - The `mmap()` system call
Suggestions on Choosing Usage Models

• Choose native execution if
  o Highly parallel code.
  o Want to use coprocessors as independent compute nodes.

• Choose AO when
  o A sufficient Byte/FLOP ratio makes offload beneficial.
  o Level-3 BLAS functions: ?GEMM, ?TRMM, ?TRSM.
  o LU, Cholesky and QR factorization
  o More in upcoming releases

• Choose CAO when either
  o There is enough computation to offset data transfer overhead.
  o Transferred data can be reused by multiple operations

• You can always run on the host if offloading does not achieve better performance
**S/D GEMM on Intel® Xeon Phi™**

Matrix Multiply Performance using Intel® Math Kernel Library on Intel® Xeon Phi™ Coprocessor SE10 and Intel® Xeon® Processor E5-2680

**SGEMM**

- Native Execution on Intel® Xeon Phi™ Coprocessor SE10
- Intel® Xeon® Processor E5-2680

**DGEMM**

- Native Execution on Intel® Xeon Phi™ Coprocessor SE10
- Intel® Xeon® Processor E5-2680

Approximately 3x performance on Intel® Xeon Phi™ coprocessor over Intel® Xeon® processor

---

**Optimization Notice**

Configuration Info - Software Versions: Intel® Math Kernel Library (Intel® MKL) 11.0.1, Intel® Manycore Platform Software Stack (MPSS) 2.1.4346; Hardware: Crown Pass Softwae Development System, Intel® Xeon® Processor E5-2680, 2 Eight-Core CPUs (20MB LLC, 2.7GHz), 32GB DDR3 RAM (1333MHz); Intel® Xeon Phi™ Coprocessor SE10, 61 cores (30.5MB total cache, 1.1GHz), 8GB GDDR5 Memory; Operating System: RHEL 6.1 CA x86_64.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmarked Source: Intel Corporation. November 2012.

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.
LU factorization on Intel® Xeon Phi™

LU Factorization Performance using Intel® Math Kernel Library
on Intel® Xeon Phi™ Coprocessor SE10 and Intel® Xeon® Processor E5-2680

Over 2.5x peak Intel® Xeon Phi™ coprocessor performance advantage over Intel® Xeon® processor

Configuration Info - Software Versions: Intel® Math Kernel Library (Intel® MKL) 11.0.1, Intel® Manycore Platform Software Stack (MPSS) 2.1.4340; Hardware: Crown Pass Software Development System, Intel® Xeon® Processor E5-2680, 2 Eight-Core CPUs (20MB LLC, 2.7GHz), 32GB DDR3 RAM (1333MHz), Intel® Xeon Phi™ Coprocessor SE10, Step B1, 61 cores (30.5MB total cache, 1.1GHz), 8GB DDR5 Memory; Operating System: RHEL 6.1 GA x86_64.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation. November 2012.

Optimization Notice: Intel's compilers may or may not optimize to the same degree as Intel microprocessors or other processors manufactured by Intel. 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.
Cholesky performance on Intel® Xeon Phi™

Cholesky Factorization Performance using Intel® Math Kernel Library on Intel® Xeon Phi™ Coprocessor SE10 and Intel® Xeon® Processor E5-2680

Over 2x peak Intel® Xeon Phi™ coprocessor performance advantage over Intel® Xeon® processor

Configuration Info - Software Versions: Intel® Math Kernel Library (Intel® MKL) 11.0.1, Intel® Manycore Platform Software Stack (MPSS) 2.1.4348; Hardware: Crown Pass Software Development System, Intel® Xeon® Processor E5-2680, 2 Eight-Core CPUs (20MB LLC, 2.7GHz), 32GB DDR3 RAM (1333MHz); Intel® Xeon Phi™ Coprocessor SE10, Step B1, 61 cores (30.5MB total cache, 1.1GHz), 8GB DDR3 Memory; Operating System: RHEL 6.1 GA x86_64.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. © Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation. November 2012.

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