



Master Informatics Eng.

2017/18

A.J.Proença

Intel MIC processors (Xeon Phi) (most slides are borrowed)

AJProença, Advanced Architectures, MiEI, UMinho, 2017/18

1

China Pulls Ahead of U.S. in Latest TOP500 List

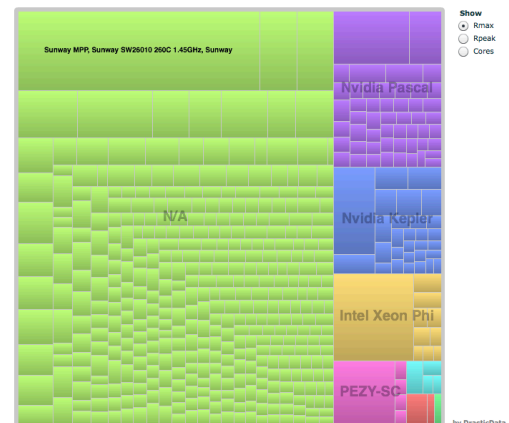
TOP500 News Team | November 13, 2017 08:59 CET

New!



The fiftieth TOP500 list of the fastest supercomputers in the world has China overtaking the US in the total number of ranked systems by a margin of 202 to 143. It is the largest number of supercomputers China has ever claimed on the TOP500 ranking, with the US presence shrinking to its lowest level since the list's inception 25 years ago.

	NAME	COUNTRY	R _{MAX} PFLOP/S	POWER MW
1	Sunway TaihuLight	China	93.0	15.4
2	Tianhe-2 (Milkyway-2)	China	33.9	17.8
3	Piz Daint	Switzerland	19.6	2.27
4	Gyokou	Japan	19.1	1.35
5	Titan	USA	17.6	8.2



A total of 102 systems employ accelerator/coprocessor technology, compared to 91 six months ago. 86 of these use NVIDIA GPUs, 12 systems make use Intel Xeon Phi coprocessor technology, and 5 are using PEZY Computing accelerators. Two systems use a combination of NVIDIA GPU and Intel Xeon Phi coprocessors. An additional 14 systems now use Xeon Phi chips as the main processing unit.

AJProença, Advanced Architectures, MiEI, UMinho, 2017/18

2



Programming Models for Intel® Xeon® processors and Intel® Many Integrated Core (Intel MIC) Architecture

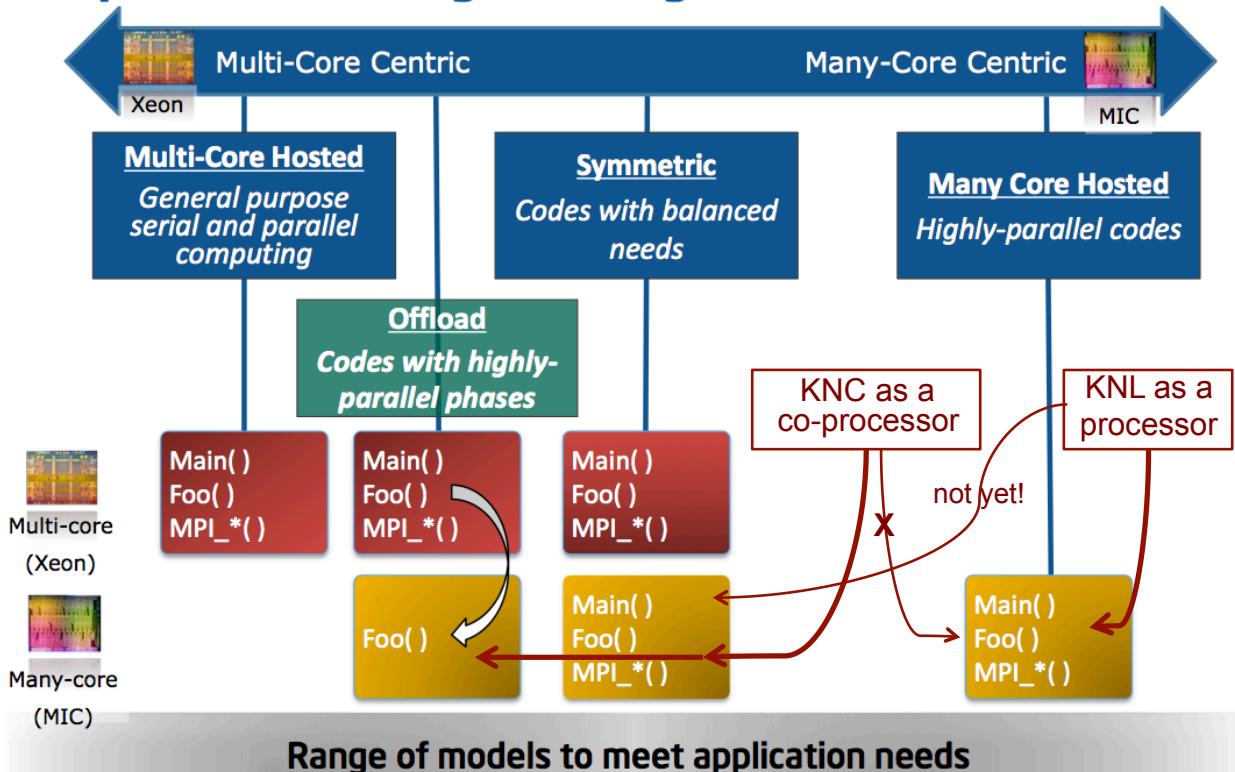
Scott McMillan

Senior Software Engineer
Software & Services Group

April 11, 2012

TACC-Intel Highly Parallel
Computing Symposium

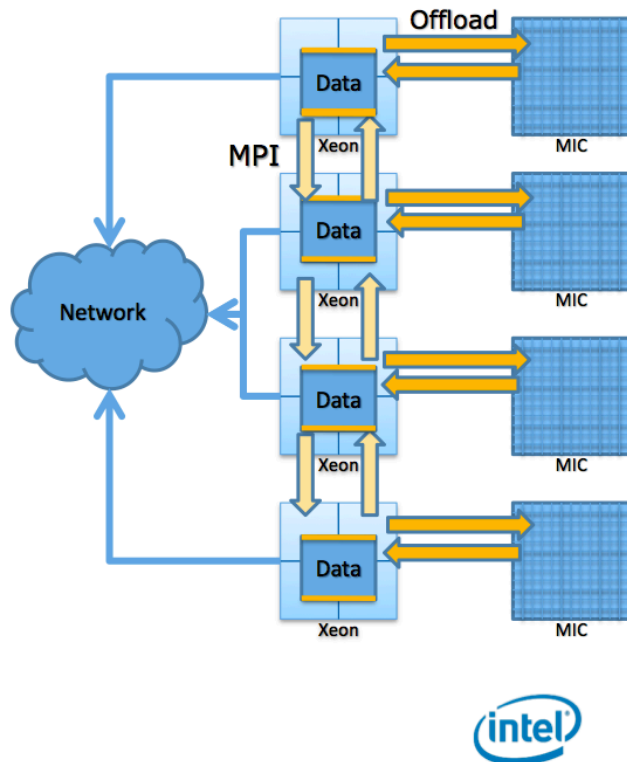
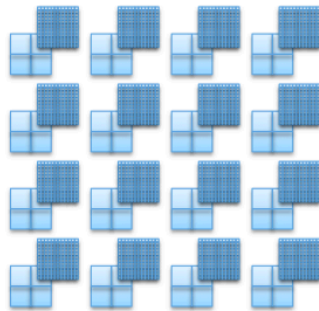
Spectrum of Programming Models and Mindsets



Programming Intel® MIC-based Systems

MPI+Offload

- MPI ranks on Intel® Xeon® processors (only)
- All messages into/out of processors
- Offload models used to accelerate MPI ranks
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* within Intel® MIC
- Homogenous network of hybrid nodes:



Offload Code Examples (KNC)

• C/C++ Offload Pragma

```
#pragma offload target (mic)
#pragma omp parallel for reduction(+:pi)
for (i=0; i<count; i++) {
    float t = (float)((i+0.5)/count);
    pi += 4.0/(1.0+t*t);
}
pi /= count;
```

• Function Offload Example

```
#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, &N,
&alpha, A, &N, B, &N, &beta, C, &N);
```

• Fortran Offload Directive

```
!dir$ omp offload target(mic)
!$omp parallel do
do i=1,10
    A(i) = B(i) * C(i)
enddo
```

• C/C++ Language Extension

```
class _Cilk_Shared common {
    int data1;
    char *data2;
    class common *next;
    void process();
};

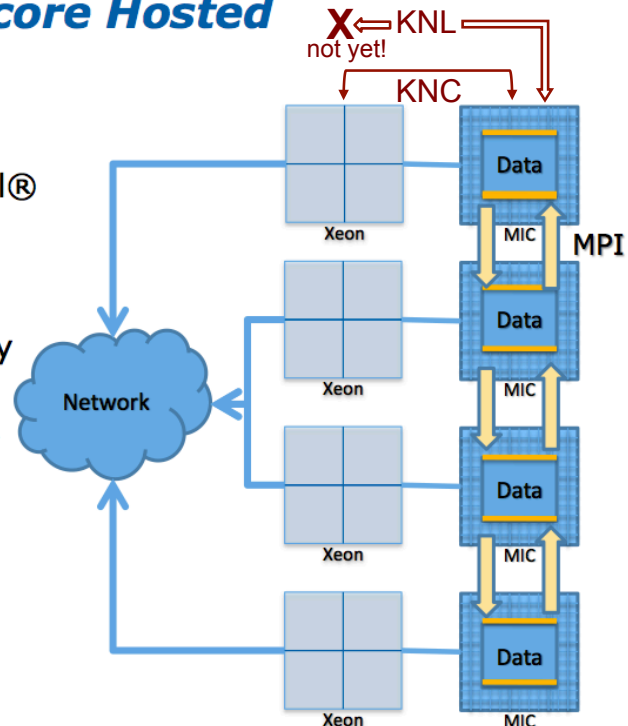
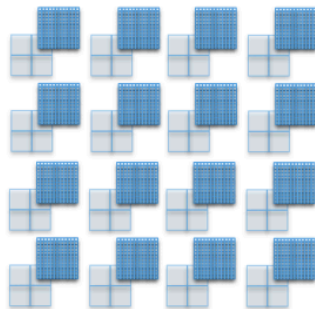
_Cilk_Shared class common obj1, obj2;
_Cilk_spawn _Offload obj1.process();
_Cilk_spawn obj2.process();
```



Programming Intel® MIC-based Systems

Many-core Hosted

- MPI ranks on Intel® MIC (only)
- All messages into/out of Intel® MIC
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads used directly within MPI processes
- Programmed as homogenous network of many-core CPUs:



13

Stand-alone Example: Computing Pi

```
# define NSET 1000000
int main ( int argc, const char** argv )
{
    long int i;
    float num_inside, Pi;
    num_inside = 0.0f;
    #pragma omp parallel for reduction(+:num_inside)
    for( i = 0; i < NSET; i++ )
    {
        float x, y, distance_from_zero;
        // Generate x, y random numbers in [0,1)
        x = float(rand()) / float(RAND_MAX + 1);
        y = float(rand()) / float(RAND_MAX + 1);
        distance_from_zero = sqrt(x*x + y*y);
        if ( distance_from_zero <= 1.0f )
            num_inside += 1.0f;
    }
    Pi = 4.0f * ( num_inside / NSET );
    printf("Value of Pi = %f \n",Pi);
}
```

Original Source Code
Compiler command line switch targets platform



Co-Processing Example: Computing Pi

```
# define NSET 1000000
int main ( int argc, const char** argv )
{   long int i;
    float num_inside, Pi;
    num_inside = 0.0f;
    #pragma offload target (MIC)
    #pragma omp parallel for reduction(+:num_inside)
    for( i = 0; i < NSET; i++ )
    {   float x, y, distance_from_zero;
        // Generate x, y random numbers in [0,1)
        x = float(rand()) / float(RAND_MAX + 1);
        y = float(rand()) / float(RAND_MAX + 1);
        distance_from_zero = sqrt(x*x + y*y);
        if ( distance_from_zero <= 1.0f )
            num_inside += 1.0f;
    }
    Pi = 4.0f * ( num_inside / NSET );
    printf("Value of Pi = %f \n",Pi);
}
```

A one line change from the CPU version



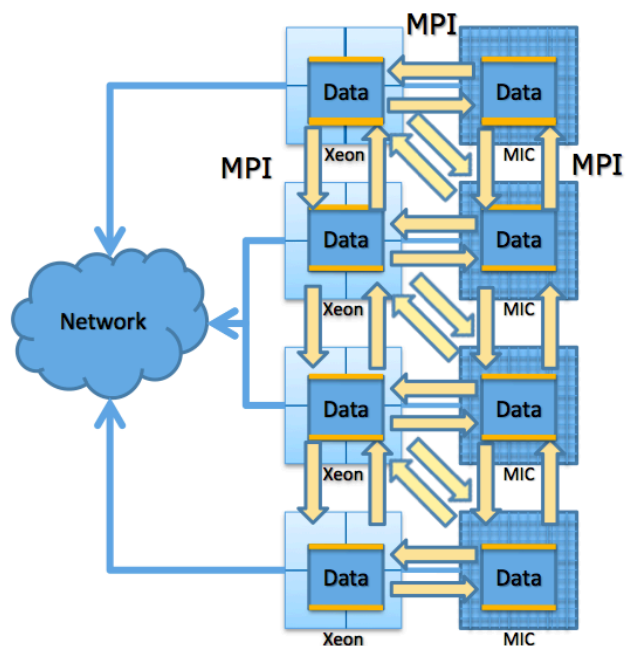
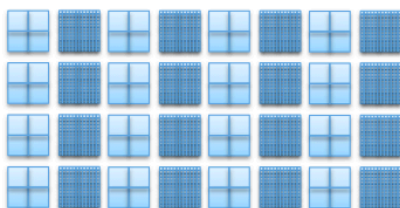
3/13/2012

Copyright © 2012, Intel Corporation. All rights reserved.

22

Programming Intel® MIC-based Systems *Symmetric*

- MPI *ranks* on Intel® MIC and Intel® Xeon® processors
- Messages to/from any core
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* used directly within MPI processes
- Programmed as heterogeneous network of homogeneous nodes:



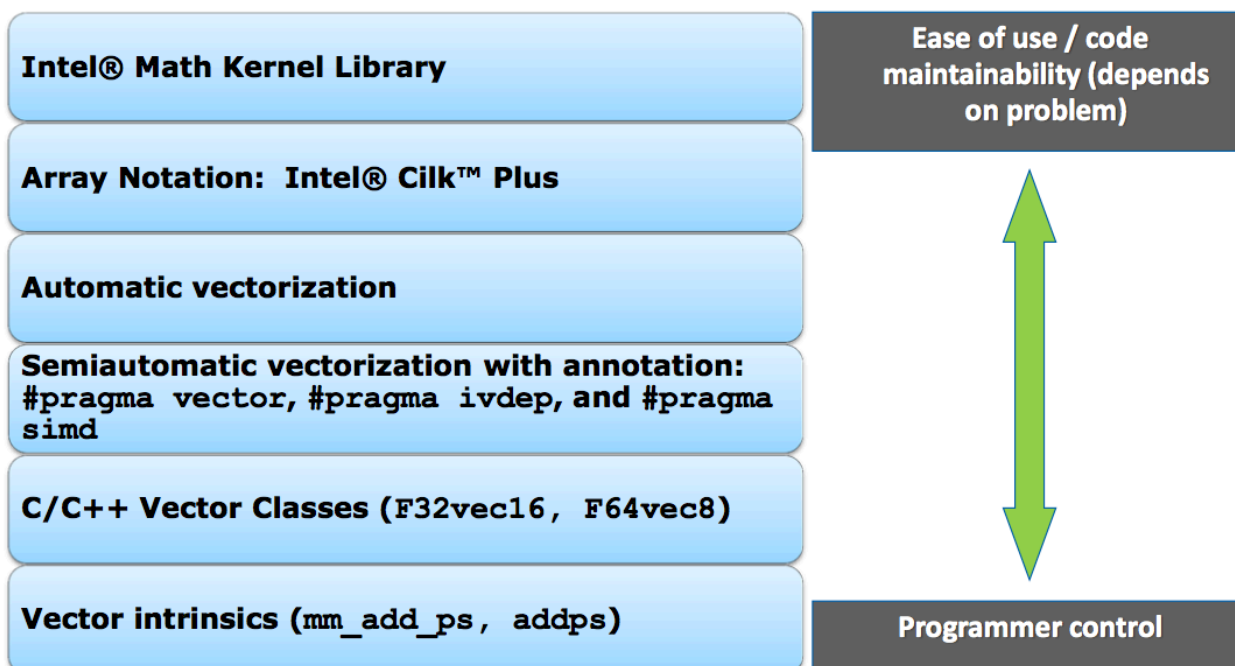
Keys to Productive Performance on Intel® MIC Architecture

- Choose the right Multi-core centric or Many-core centric model for your application
- Vectorize your application (today)
 - Use the Intel vectorizing compiler
- Parallelize your application (today)
 - With MPI (or other multi-process model)
 - With threads (via Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads, etc.)
- Go asynchronous to overlap computation and communication



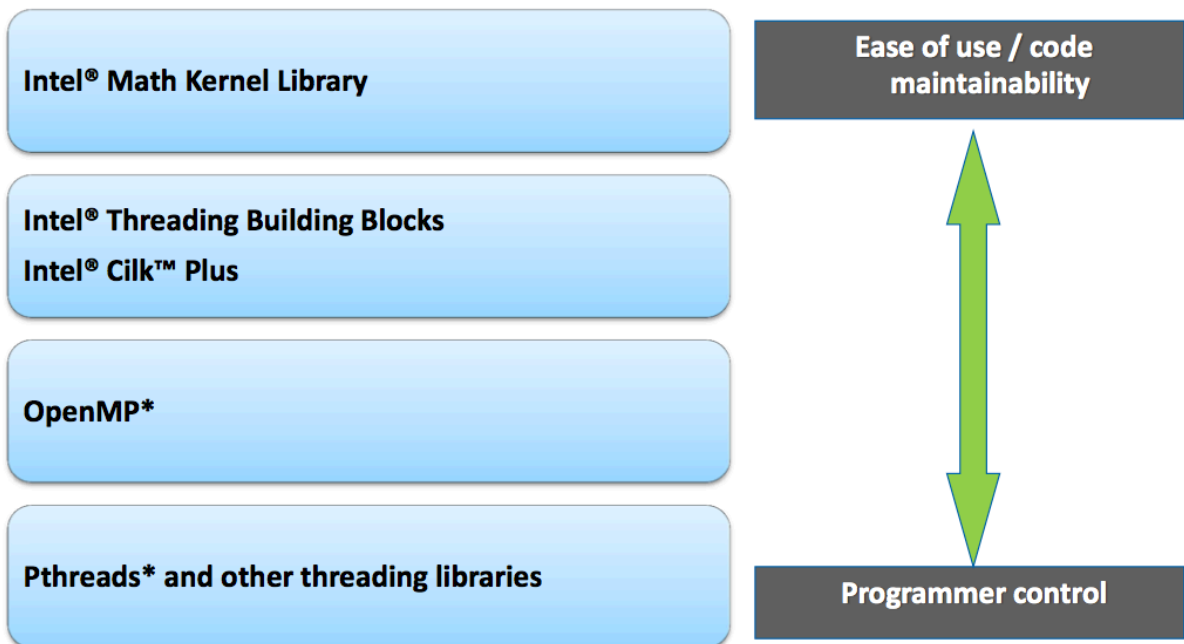
15

Options for Vectorization



17

Options for Thread Parallelism



16

Summary

- Intel® MIC Architecture offers familiar and flexible programming models
- Hybrid MPI/threading is becoming increasingly important as core counts grow
- Intel tools support hybrid programming today, exploiting existing standards
- Hybrid parallelism on Intel® Xeon® processors + Intel® MIC delivers superior productivity through code reuse
- Hybrid programming today on Intel® Xeon® processors readies you for Intel® MIC



19

INTRODUCTION TO THE INTEL® XEON PHI™ PROCESSOR (CODENAME “KNIGHTS LANDING”)

Dr. Harald Servat - HPC Software Engineer
Data Center Group – Innovation Performing and Architecture Group

Summer School in Advanced Scientific Computing 2016
February 21st, 2016 – Braga, Portugal
June

INTEL® XEON PHI™ PROCESSOR FAMILY ARCHITECTURE OVERVIEW

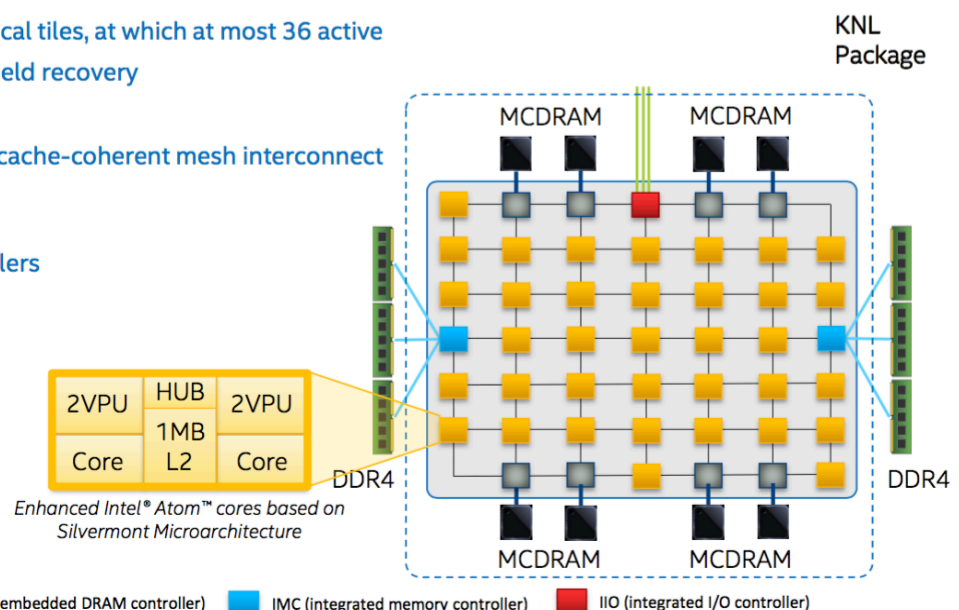
Codenamed “Knights Landing” or KNL

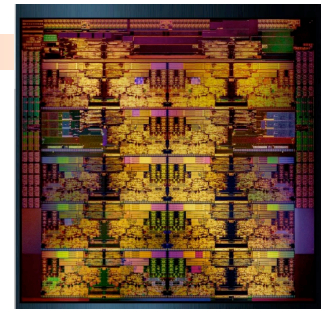
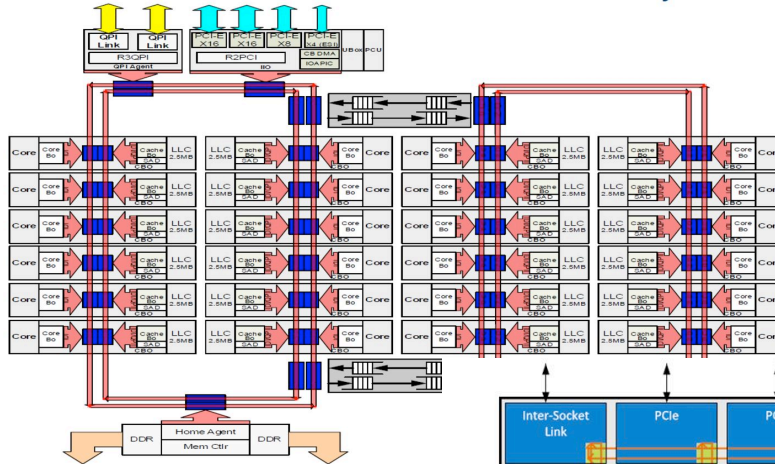
Comprises 38 physical tiles, at which at most 36 active

- Remaining for yield recovery

Introduces new 2D cache-coherent mesh interconnect (Untile)

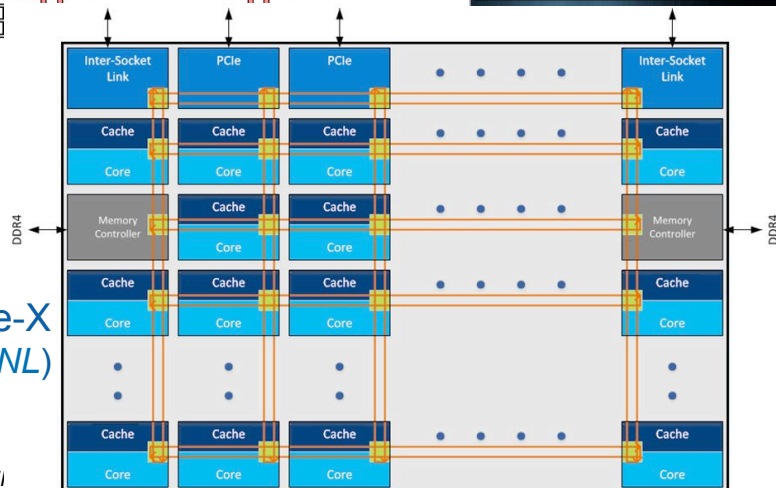
- Tiles
- Memory controllers
- I/O controllers
- Other agents





Intel 18-core Skylake-X (follows KNL)

AJProença, Advanced Architectures, MiEi



KNL PROCESSOR TILE

Tile

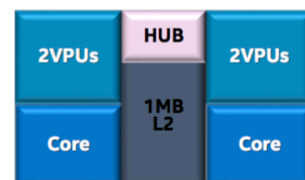
- 2 cores, each with 2 vector processing units (VPU)
- 1 MB L2-cache shared between the cores

Core

- Binary compatible with Xeon
- Enhanced Silvermont (Atom)-based for HPC w/ 4 threads
- Out-of-order core
- 2-wide decode, 6-wide execute (2 int, 2 fp, 2 mem), 2-wide retire

2 VPU

- 512-bit SIMD (AVX512) 32SP/16DP per unit
- Legacy X87, SSE, AVX and AVX2 support



KNIGHTS LANDING VS. KNIGHTS CORNER FEATURE COMPARISON

FEATURE	INTEL® XEON PHI™ COPROCESSOR 7120P	KNIGHTS LANDING PRODUCT FAMILY
Processor Cores	Up to 61 enhanced P54C Cores	Up to 72 enhanced Silvermont cores
Key Core Features	In order 4 threads / core (back-to-back scheduling restriction) 2 wide	Out of order 4 threads / core 2 wide
Peak FLOPS ¹	SP: 2.416 TFLOPs • DP: 1.208 TFLOPs	Up to 3x higher
Scalar Performance ¹	1X	Up to 3x higher
Vector ISA	x87, (no Intel® SSE or MMX™), Intel IMIC	x87, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, Intel® AVX, AVX2, AVX-512 (no Intel® TSX)
Interprocessor Bus	Bidirectional Ring Interconnect	Mesh of Rings Interconnect

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. See benchmark tests and configurations in the speaker notes. For more information go to <http://www.intel.com/performance>

¹- Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.



20

TAKING BENEFIT OF THE CORE

Threading

- Ensure that thread affinities are set.
- Understand affinity and how it affects your application (i.e. which threads share data?).
- Understand how threads share core resources.
 - An individual thread has the highest performance when running alone in a core.
 - Running 2 or 4 threads in a core may result in higher per core performance but lower per thread performance.
 - Due to resource partitioning, 3 thread configuration will have fewer aggregative resources than 1, 2 or 4 threads per core. 3 threads in a core is unlikely to perform better than 2 or 4 threads.

Vectorization

- Prefer AVX512 instructions and avoid mixing SSE, AVX and AVX512 instructions.
- Avoid cache-line splits; align data structures to 64 bytes.
- Avoid gathers/scatters; replace with shuffles/permutates for known sequences.
- Use hardware transcendentals (fast-math) whenever possible.
- AVX512 achieves best performance when not using masking
- KNC intrinsic code is unlikely to generate optimal KNL code, recompile from HL language.

OpenMP

From Wikipedia, the free encyclopedia

Thread affinity [\[edit \]](#)

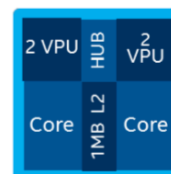
Some vendors recommend setting the [processor affinity](#) on OpenMP threads to associate them with particular processor cores. [\[33\]\[34\]\[35\]](#) This minimizes thread migration and context-switching cost among cores. It also improves the data locality and reduces the cache-coherency traffic among the cores (or processors).



28

DATA LOCALITY: NESTED PARALLELISM

- Recall that KNL cores are grouped into tiles, with two cores sharing an L2.
- Effective capacity depends on locality:
 - 2 cores sharing no data => 2 x 512 KB
 - 2 cores sharing all data => 1 x 1 MB
- Ensuring good locality (e.g. through blocking or nested parallelism) is likely to improve performance.



```
#pragma omp parallel for num_threads(ntiles)
for (int i = 0; i < N; ++i)
{
    #pragma omp parallel for num_threads(8)
    for (int j = 0; j < M; ++j)
    {
        ...
    }
}
```



29

KNL PROCESSOR UNTILE

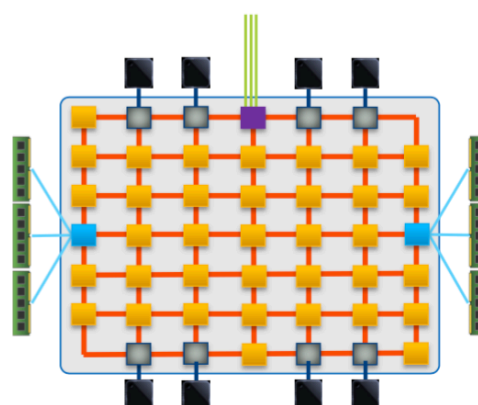
Comprises a mesh connecting the tiles (in red) with the MCDRAM and DDR memories.

- Also with I/O controllers and other agents

Caching Home Agent (CHA) holds portion of the distributed tag directory and serves as connection point between tile and mesh

- No L3 cache as in Xeon

Cache coherence uses MESIF protocol (Modified, Exclusive, Shared, Invalid, Forward)

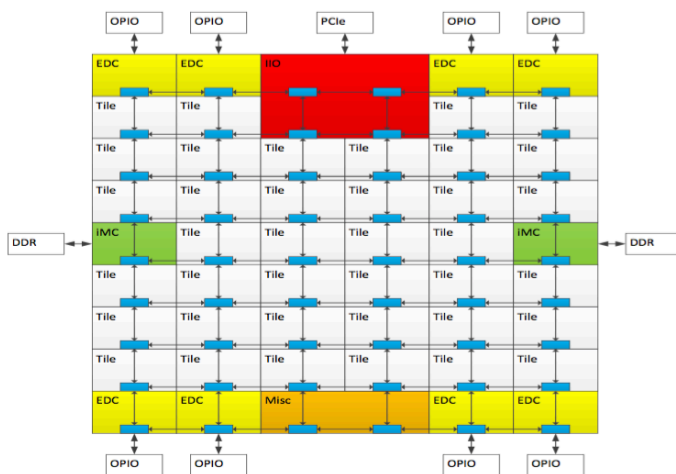


■ Tile
 ■ EDC (embedded DRAM controller)
 ■ IMC (integrated memory controller)
 ■ IIO (integrated I/O controller)



31

KNL MESH INTERCONNECT



Mesh of Rings

- Every row and column is a ring
- YX routing: Go in Y → Turn → Go in X
 - 1 cycle to go in Y, 2 cycles to go in X
- Messages arbitrate at injection and on turn

Mesh at fixed frequency of 1.7 GHz

Distributed Directory Coherence protocol

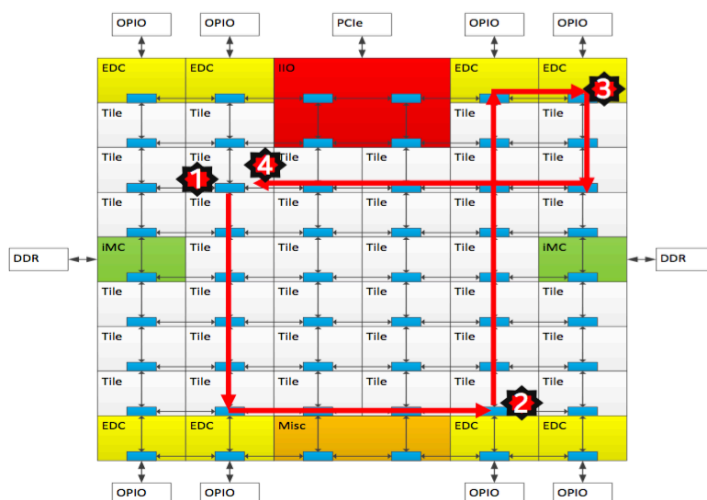
KNL supports Three Cluster Modes

- 1) All-to-all
- 2) Quadrant
- 3) Sub-NUMA Clustering

Selection done at boot time.



CLUSTER MODE: ALL-TO-ALL



Address uniformly hashed across all distributed directories

No affinity between Tile, Directory and Memory

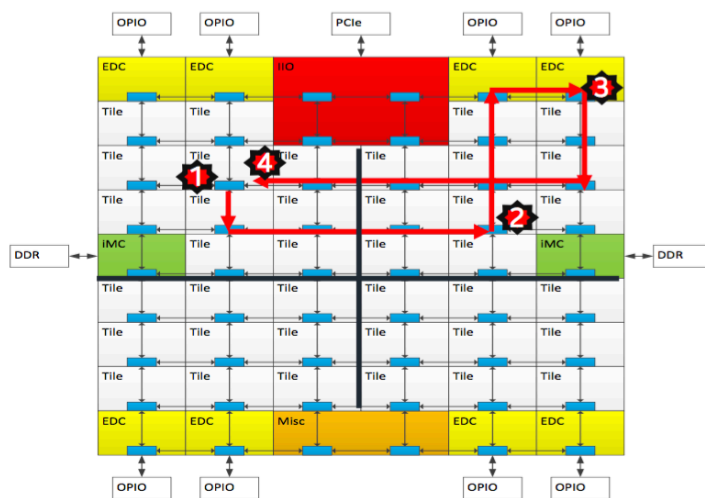
Lower performance mode, compared to other modes. Mainly for fall-back

Typical Read L2 miss

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor



CLUSTER MODE: QUADRANT



Chip divided into four Quadrants

Affinity between the Directory and Memory

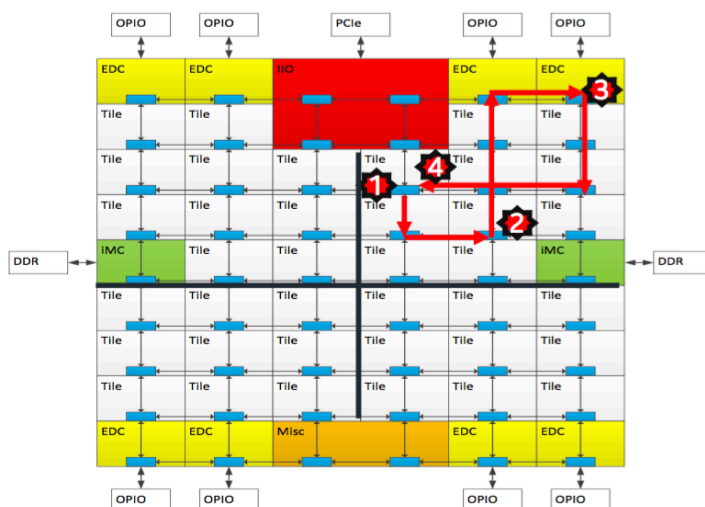
Lower latency and higher BW than all-to-all

SW Transparent

Typical Read L2 miss

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor

CLUSTER MODE: SUB-NUMA CLUSTERING (SNC4)



Each Quadrant (Cluster) exposed as a separate NUMA domain to OS

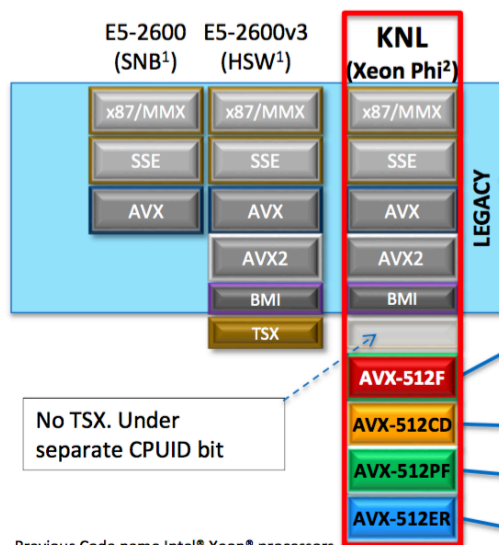
Analogous to 4-socket Xeon

SW Visible

Typical Read L2 miss

1. L2 miss encountered
2. Send request to the distributed directory
3. Miss in the directory. Forward to memory
4. Memory sends the data to the requestor

KNL HARDWARE INSTRUCTION SET



1. Previous Code name Intel® Xeon® processors
2. Xeon Phi = Intel® Xeon Phi™ processor

KNL implements all legacy instructions

- Legacy binary runs w/o recompilation
- KNC binary requires recompilation

KNL introduces AVX-512 Extensions

- 512-bit FP/Integer Vectors
- 32 registers & 8 mask registers
- Gather/Scatter

Conflict Detection: Improves Vectorization

Prefetch: Gather and Scatter Prefetch

Exponential and Reciprocal Instructions



GUIDELINES FOR WRITING VECTORIZABLE CODE

Prefer simple “for” or “DO” loops

Write straight line code. Try to avoid:

- function calls (unless inlined or SIMD-enabled functions)
- branches that can't be treated as masked assignments.

Avoid dependencies between loop iterations

- Or at least, avoid read-after-write dependencies

Prefer arrays to the use of pointers

- Without help, the compiler often cannot tell whether it is safe to vectorize code containing pointers.
- Try to use the loop index directly in array subscripts, instead of incrementing a separate counter for use as an array address.
- Disambiguate function arguments, e.g. -fargument-noalias

Use efficient memory accesses

- Favor inner loops with unit stride
- Minimize indirect addressing `a[i] = b[ind[i]]`
- Align your data consistently where possible (to 16, 32 or 64 byte boundaries)



INTEL® COMPILER SWITCHES TARGETING INTEL® AVX-512

Switch	Description
-xmic-avx512	KNL only <u>Not</u> a fat binary.
-xcore-avx512	Future Xeon only <u>Not</u> a fat binary.
-xcommon-avx512	AVX-512 subset common to both. <u>Not</u> a fat binary.
-axmic-avx512 etc.	Fat binaries. Allows to target KNL and other Intel® Xeon® processors

Don't use `-mmic` with KNL !

Best would be to use `-axcore-avx512,mic-avx512 -xcommon-avx512`

All supported in 16.0 and forthcoming 17.0 compilers

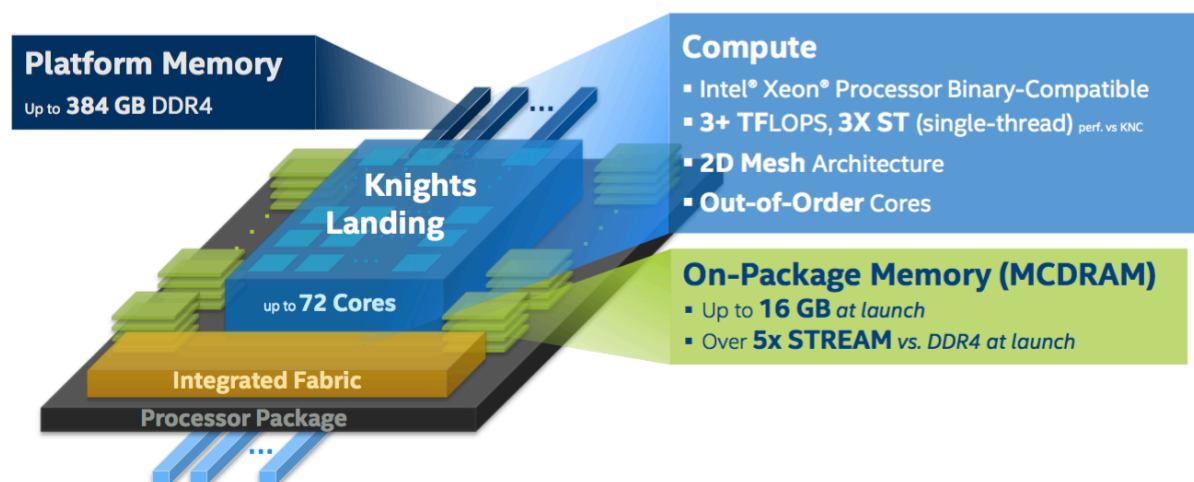
Binaries built for earlier Intel® Xeon® processors will run unchanged on KNL

Binaries built for Intel® Xeon Phi™ coprocessors will not.



49

INTEL® XEON PHI™ X200 PROCESSOR OVERVIEW



51

MCDRAM MODES

Cache mode

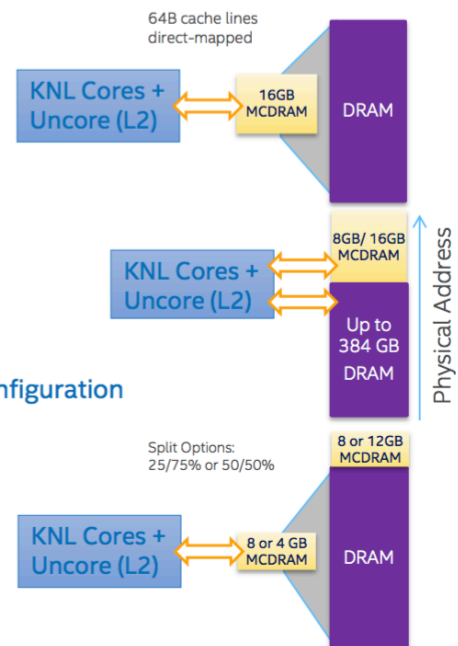
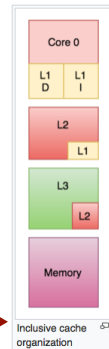
- Direct mapped cache
- Inclusive cache
- Misses have higher latency
 - Needs MCDRAM access + DDR access
- No source changes needed to use, automatically managed by hw as if LLC

Flat mode

- MCDRAM mapped to physical address space
- Exposed as a NUMA node
 - Use `numactl --hardware, lscpu` to display configuration
- Accessed through `memkind` library or `numactl`

Hybrid

- Combination of the above two
 - E.g., 8 GB in cache + 8 GB in Flat Mode



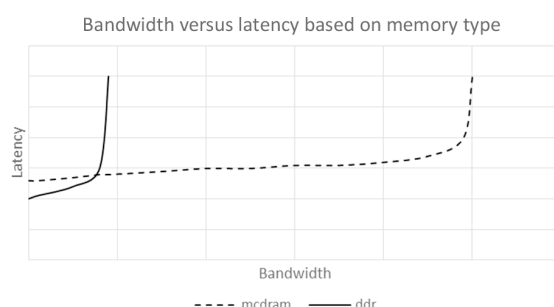
53

TAKE AWAY MESSAGE: CACHE VS FLAT MODE



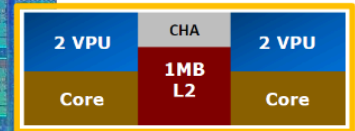
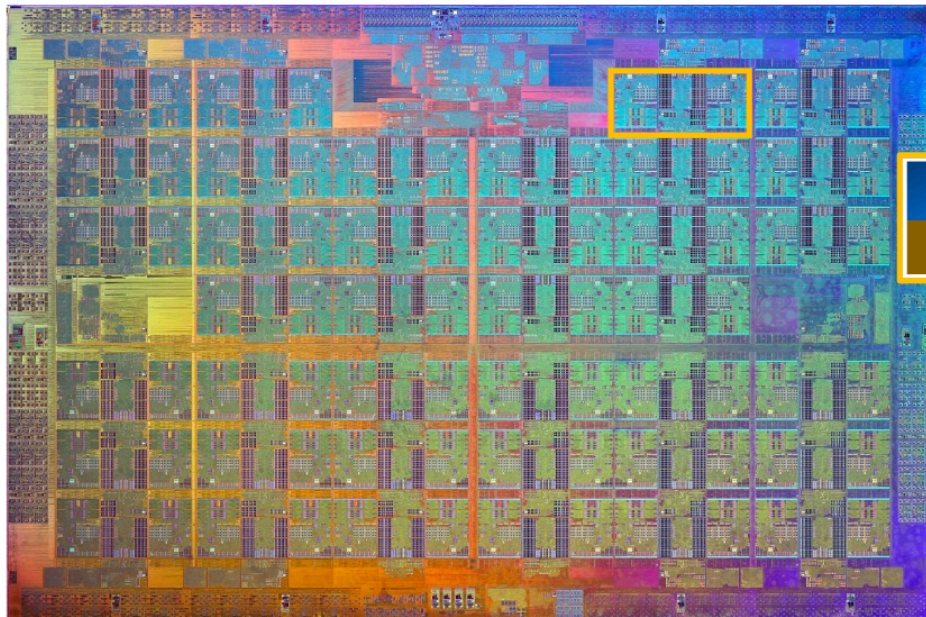
	Recommended				
	DDR Only	MCDRAM as Cache	MCDRAM Only	Flat DDR + MCDRAM	Hybrid
Software Effort	No software changes required			Change allocations for bandwidth-critical data.	
Performance	Not peak performance.		Best performance.		
			Limited memory capacity	Optimal HW utilization + opportunity for new algorithms	

s latency based on memory type



55

Intel® Knights Landing die



Note:

- 38 tiles, 76 cores
- max on sale: 36 tiles

Knights Landing products

