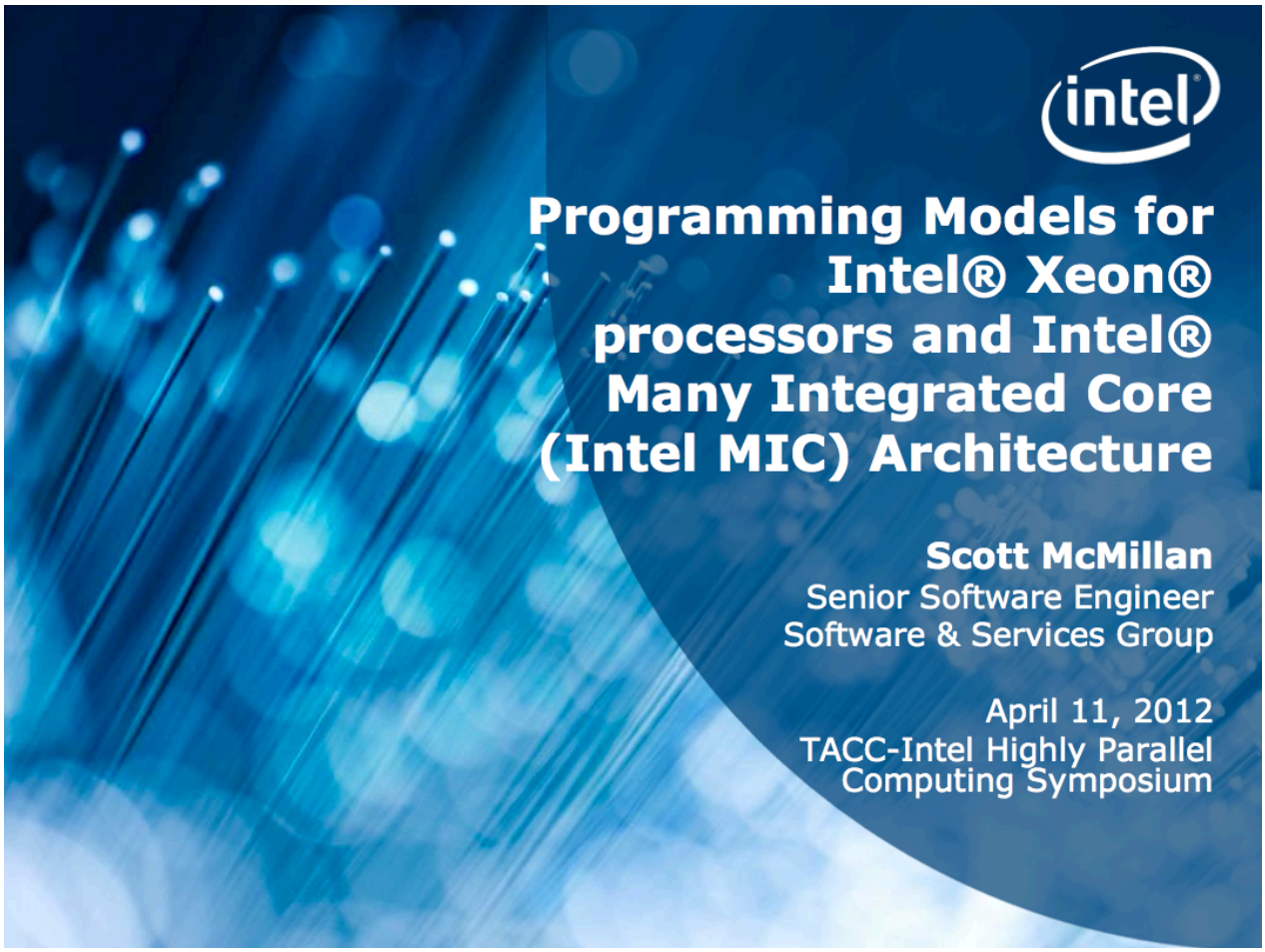
Master Informatics Eng.

2016/17

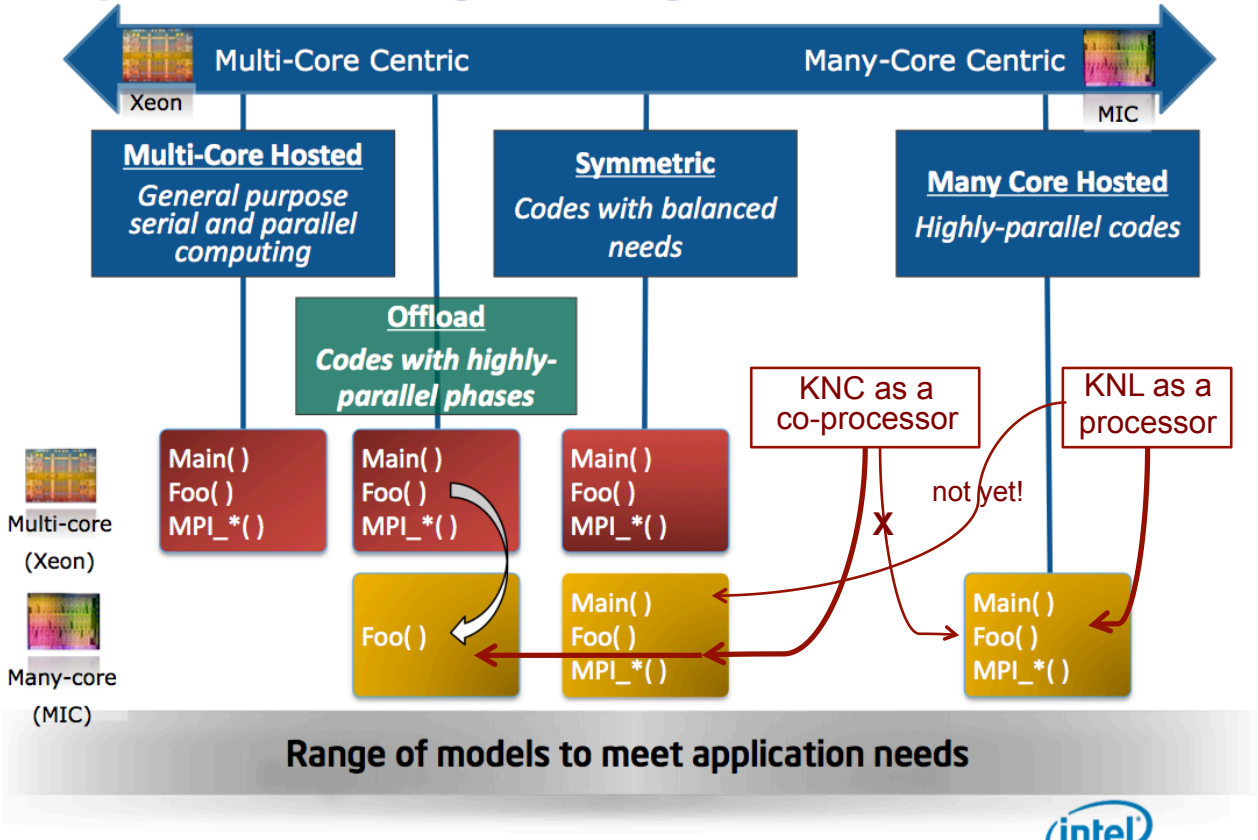
A.J.Proença

Data Parallelism 4 (*exploring many-core*)

(most slides are borrowed)

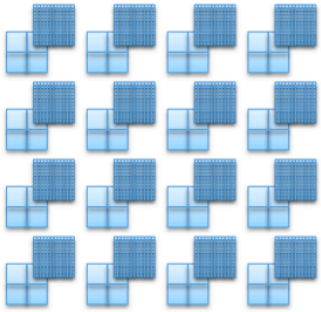


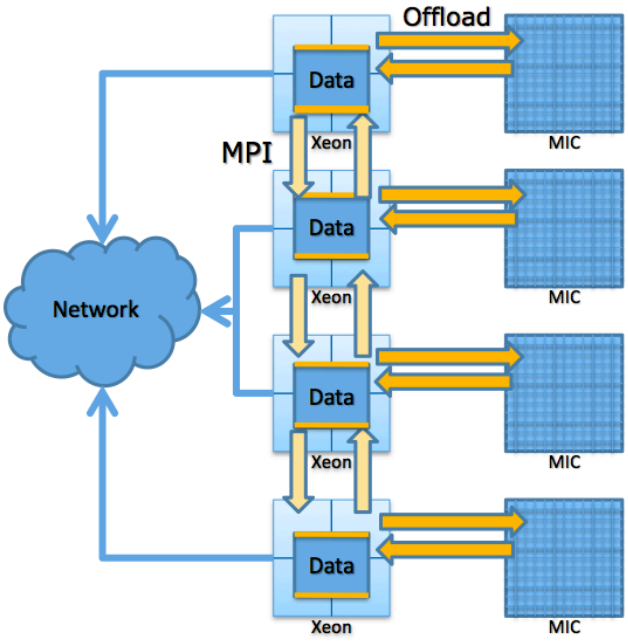
Spectrum of Programming Models and Mindsets



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

- Fortran Offload Directive

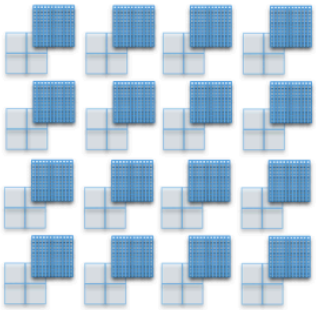
```
!dir$ omp offload target(mic)
!$omp parallel do
     do i=1,10
               A(i) = B(i) * C(i)
     enddo
```

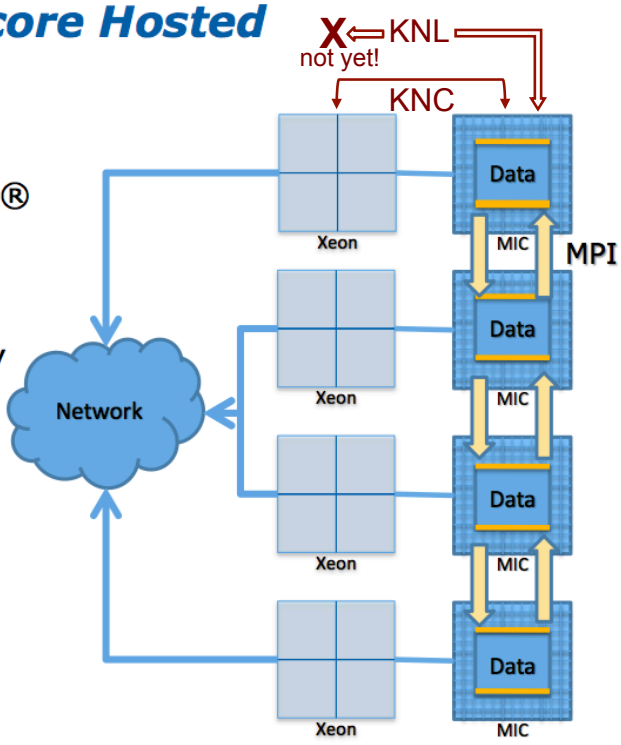
- C/C++ Language Extension

```
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:

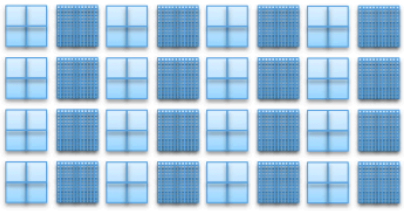


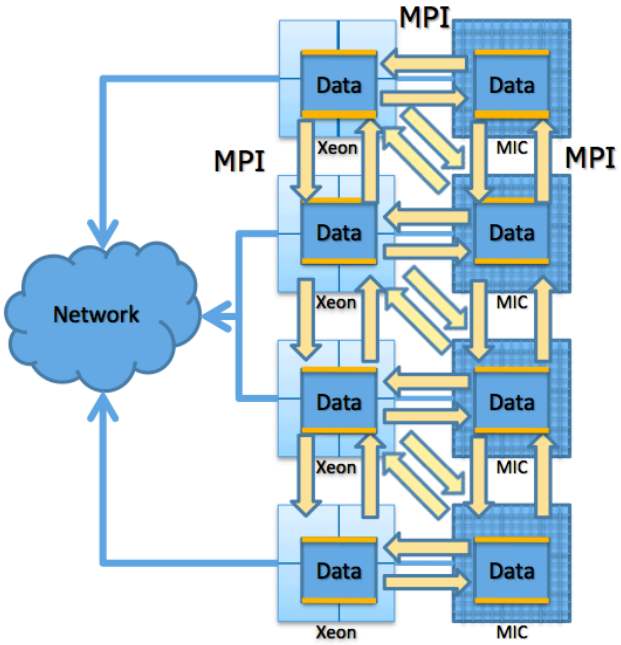


Programming Intel® MIC-based Systems
Symmetric

- 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

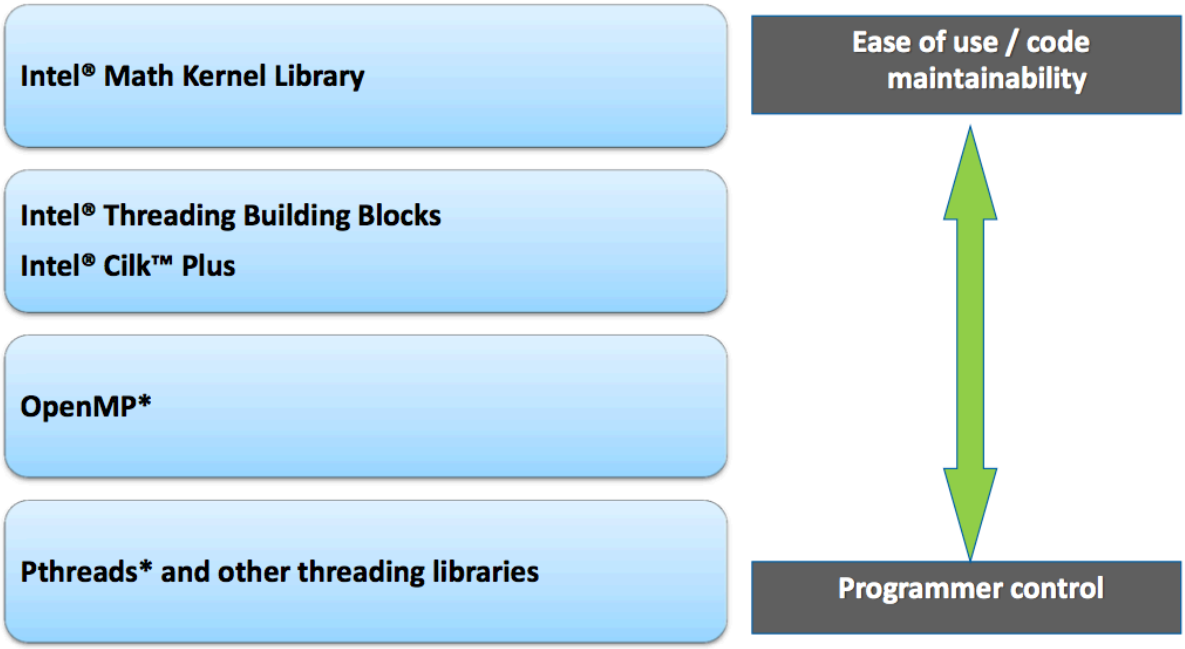
- Choose the right Multi-core centric or Many-core centric model for your application

- Vectorize your application (today)
  - Use the Intel vectorizing compiler

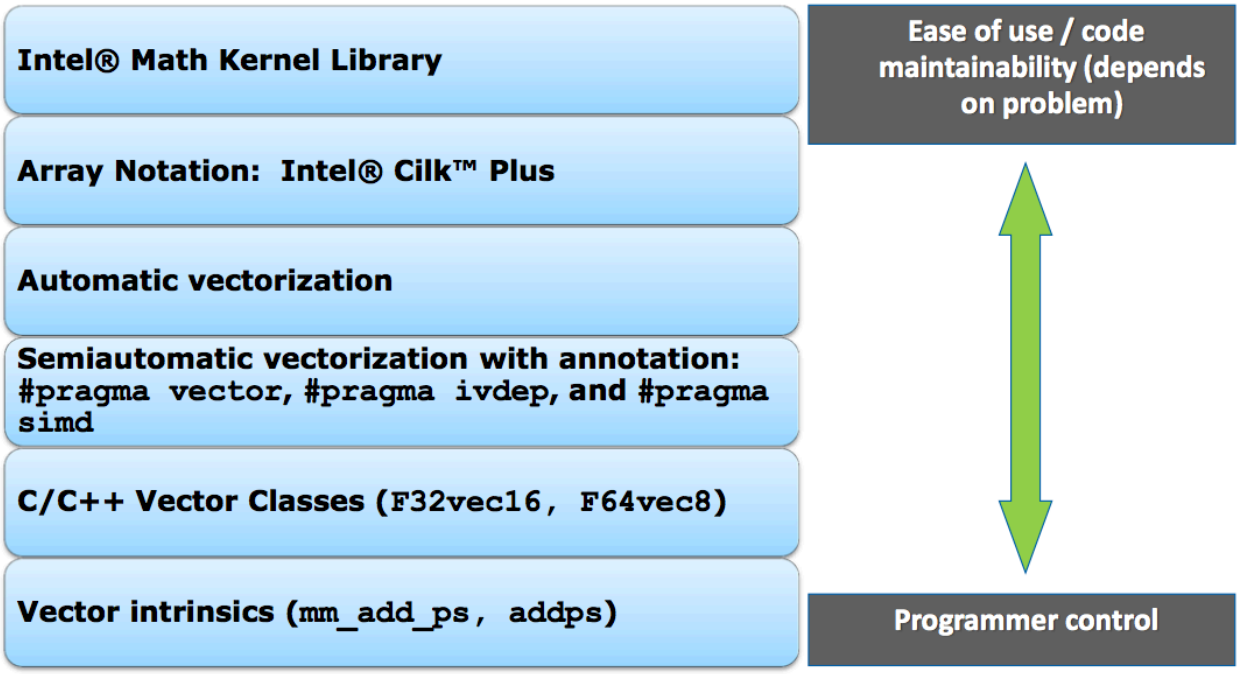
- Parallelize your application (today)
  - With MPI (or other multi-process model)
  - With threads (via Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads, etc.)

- Go asynchronous to overlap computation and communication

Options for Thread Parallelism



Options for Vectorization



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

Intel® Many Integrated Core Architecture: An Overview and Programming Models

Jim Jeffers
SW Product Application Engineer
Technical Computing Group

Sponsors of Tomorrow.™

"Stand-alone" Intel® MIC Architecture Computing Environment

- Intel® MIC Architecture software environment includes a highly functional, Linux* OS running on the co-processor with:
  - A familiar interactive shell
  - IP Addressability [headless node]
  - A local file system with subdirectories, file reads, writes, etc
  - standard i/o including printf
  - Virtual memory management
  - Process, thread management & scheduling
  - Interrupt and exception handling
  - Semaphores, mutexes, etc…
- What does this mean?
  - A large majority of existing code even with OS oriented calls like fork() can port with a simple recompile
  - Intel MIC Architecture natively supports parallel coding models like Intel® Cilk™ Plus, Intel® Threading Building Blocks, pThreads*, OpenMP*

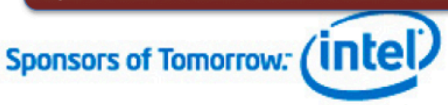
fooey.c

```
main( )
{
   printf("running Foo()\n");
   Foo( );
}

Foo()
{
   printf("fooey\n");
}
```

Intel MIC Architecture
(Knights Corner console)

```
mymic>ls
 fooey
mymic>./fooey
running Foo()
fooey
mymic>
```



Stand-alone Example: Computing Pi

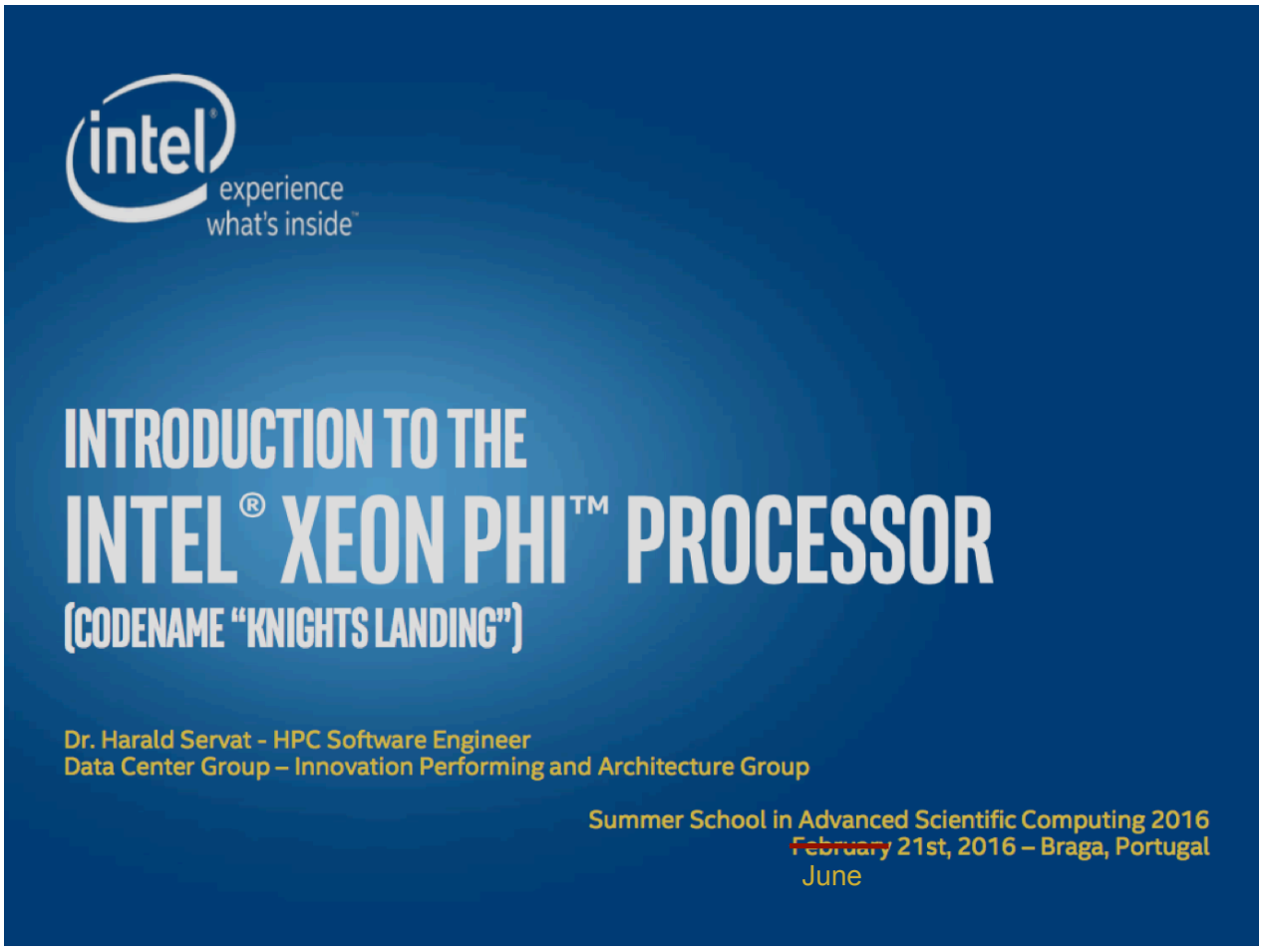
```
# define NSET 1000000
int main ( int argc, const char** argv )
{   long int i;
    float num_inside, Pi;
    num_inside = 0.0f;
    #pragma omp parallel for reduction(+:num_inside)
    for( i = 0; i < NSET; i++ )
    {      float x, y, distance_from_zero;
                    // Generate x, y random 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



INTEL® XEON PHI™ PROCESSOR FAMILY ARCHITECTURE OVERVIEW

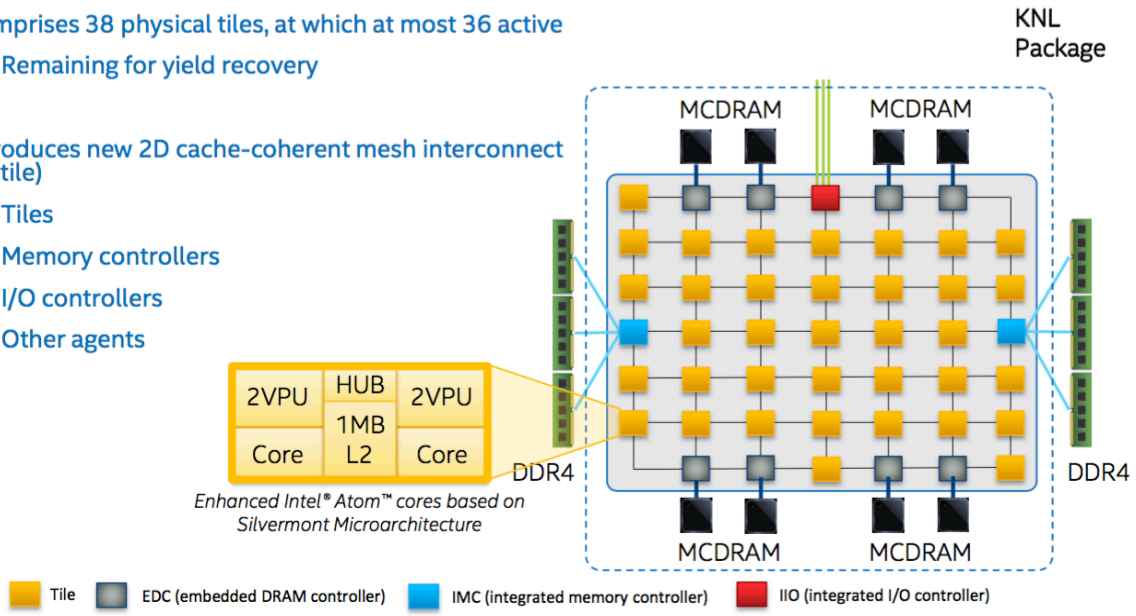
Codenamed "Knights Landing" or KNL

Comprises 38 physical tiles, at which at most 36 active

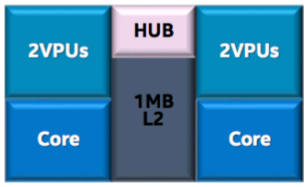
- Remaining for yield recovery

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

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



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.

TAKING BENEFIT OF THE CORE

Threading

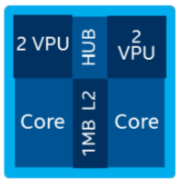
- Ensure that thread affinities are set.
- Understand affinity and how it affects your application (i.e. which threads share data?).
- Understand how threads share core resources.
  - An individual thread has the highest performance when running alone in a core.
  - Running 2 or 4 threads in a core may 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/permutes for known sequences.
- Use hardware trascendentals (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.

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)
      {
            …
      }
}
```


KNL PROCESSOR UNTILE

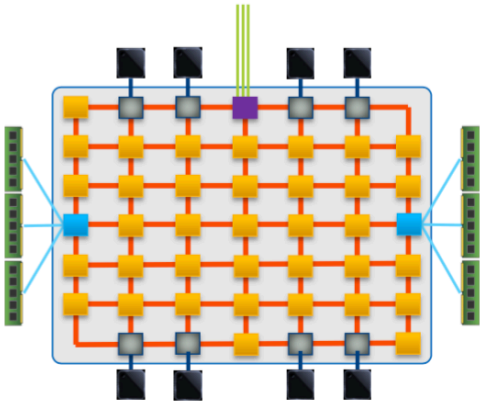
Comprises a mesh connecting the tiles (in red) with the MCDRAM and DDR memories.

- 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

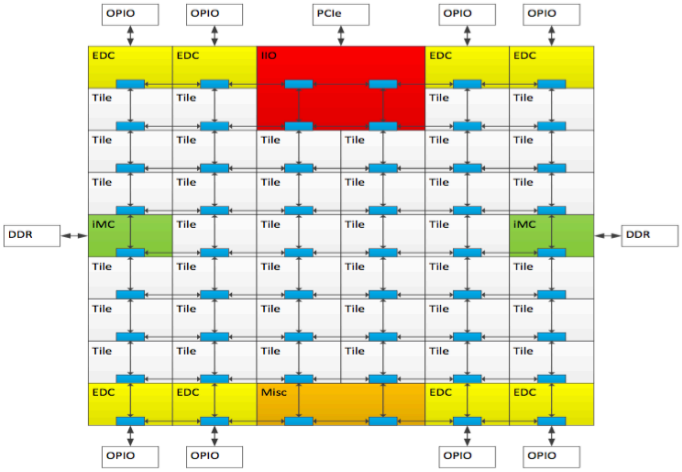
- No L3 cache as in Xeon

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



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

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

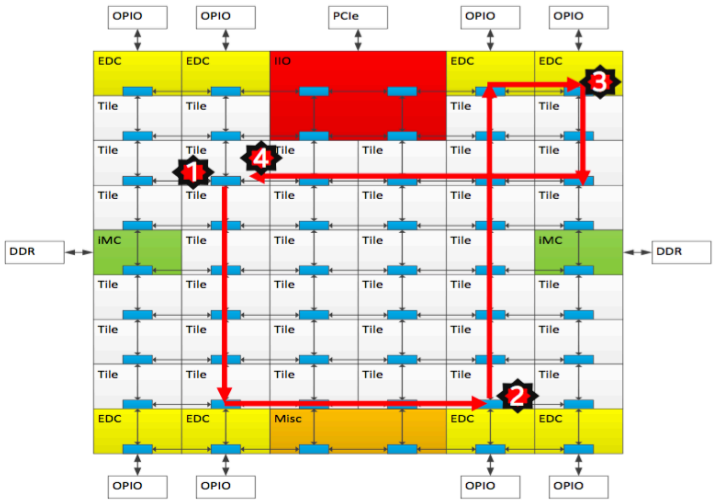
Distributed Directory Coherence protocol

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

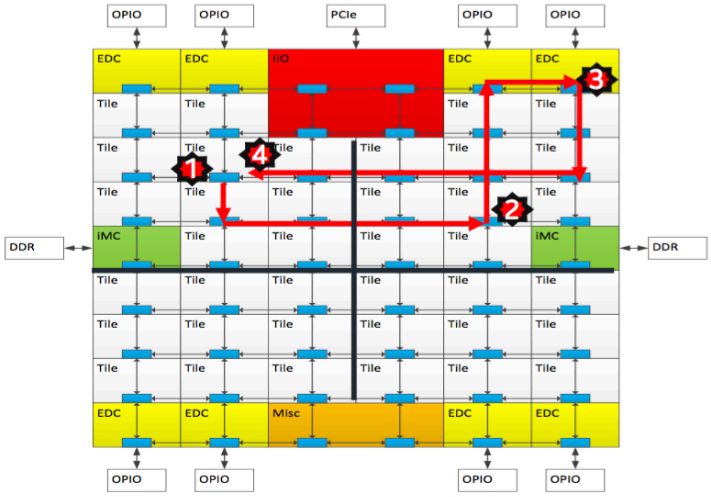
No affinity between Tile, Directory and Memory

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

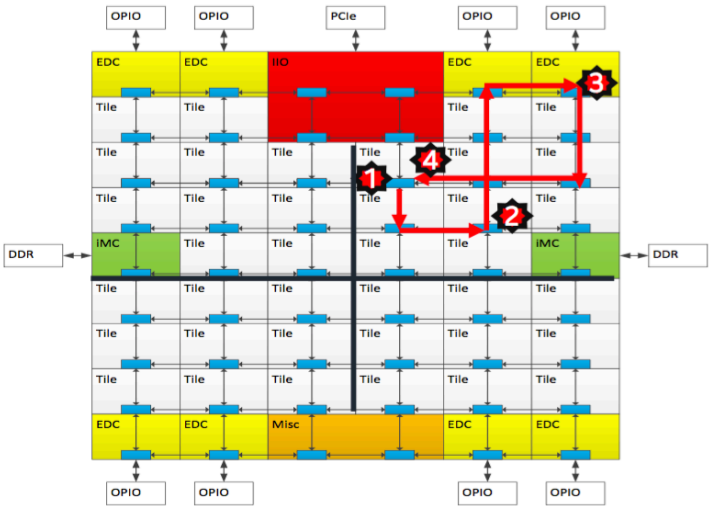
Lower latency and higher BW than all-to-all

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

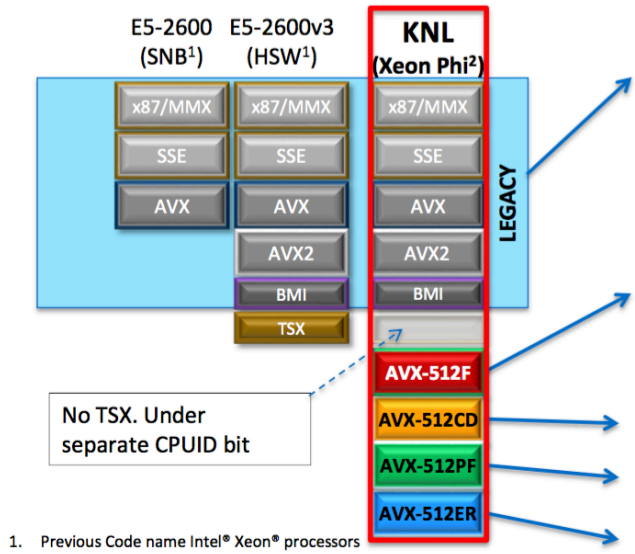
Analogous to 4-socket Xeon

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



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

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

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 Not a fat binary.
`-xcore-avx512`	Future Xeon only Not a fat binary.
`-xcommon-avx512`	AVX-512 subset common to both. Not a fat binary.
`-axmic-avx512 etc.`	Fat binaries. Allows to target KNL and other Intel® Xeon® processors

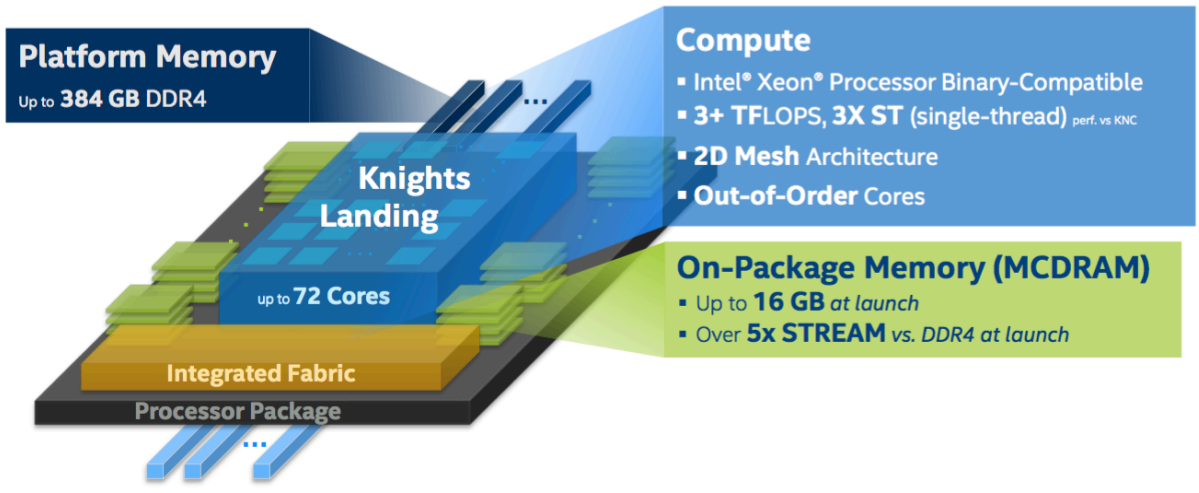
Don't use –mmic with KNL !

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.

INTEL® XEON PHI™ X200 PROCESSOR OVERVIEW

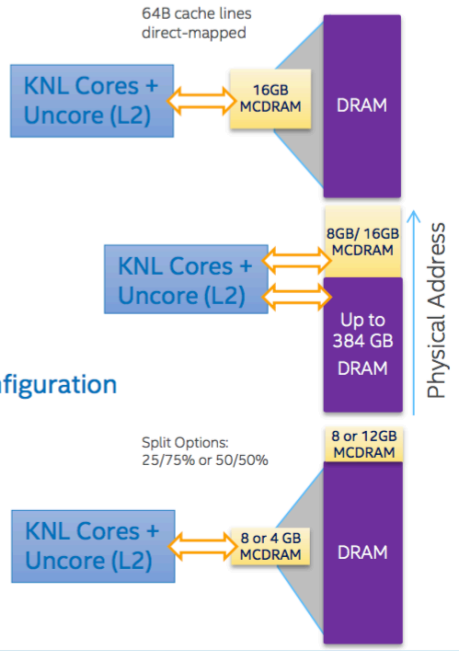


MCDRAM MODES

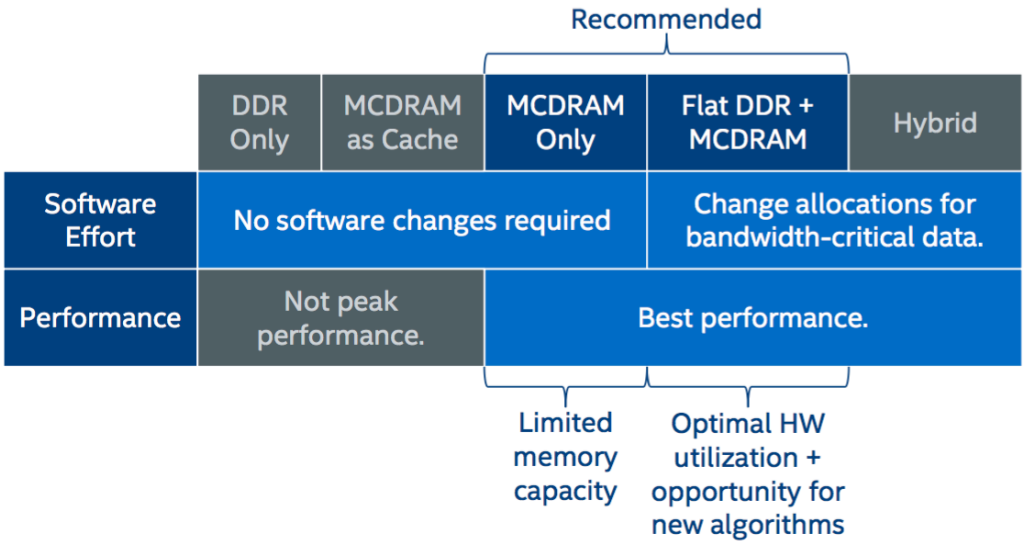
Cache mode
- Direct mapped cache
- Inclusive cache
- Misses have higher latency
  - Needs MCDRAM access + DDR access
- No source changes needed to use, automatically managed by hw as if LLC

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

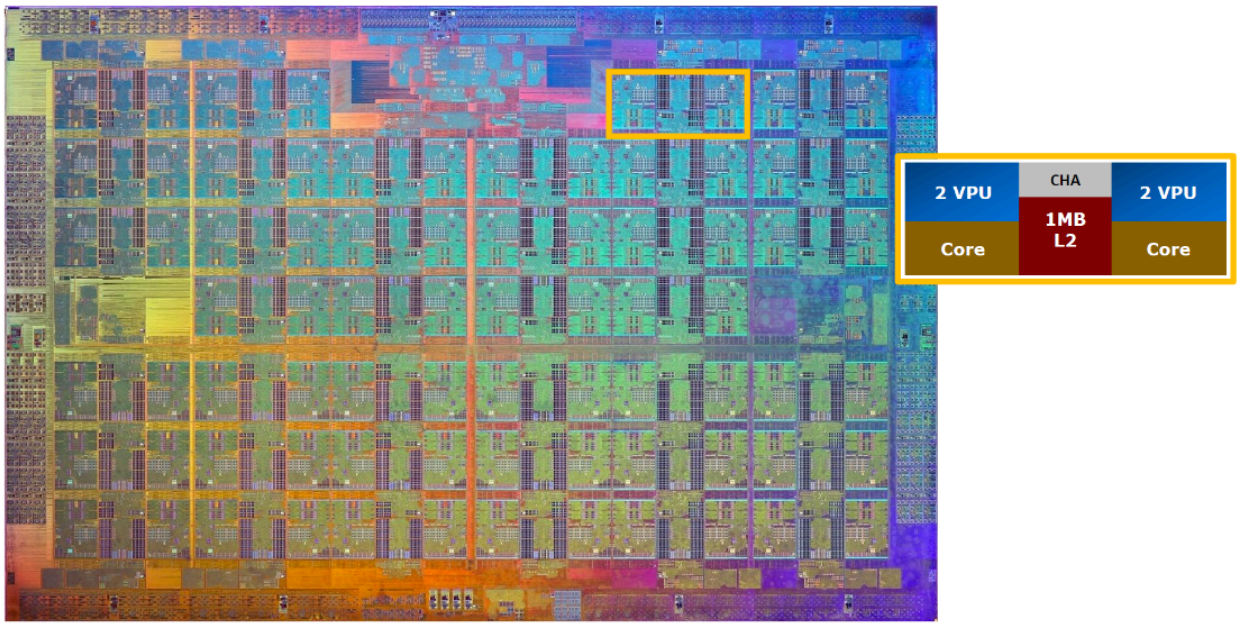
Hybrid
- Combination of the above two
  - E.g., 8 GB in cache + 8 GB in Flat Mode



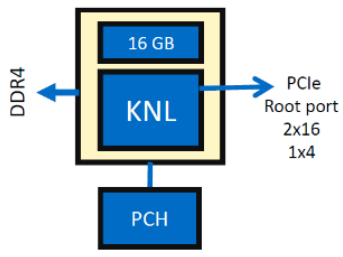
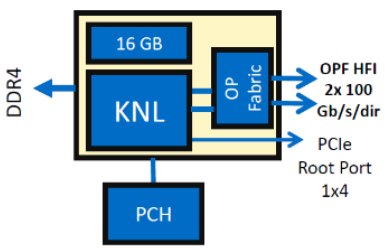
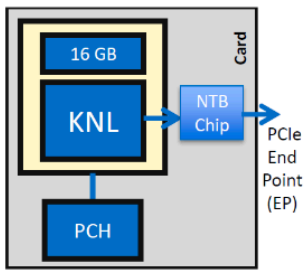
TAKE AWAY MESSAGE: CACHE VS FLAT MODE



Intel® Knights Landing die



Knights Landing products

KNL	KNL w/ Omni-Path	KNL Card
		
6 DDR channels Up to 16 GB MCDRAM 36-lanes Gen3 PCIe (root port)	6 DDR channels Up to 16 GB MCDRAM 4-lanes Gen3 PCIe (root port) Omni-Path fabric (200 Gb/s/dir)	No DDR Channels Up to 16 GB MCDRAM 16-lanes Gen3 PCIe (end point) NTB Chip to create PCIe EP
Self boot socket		PCIe Card