Intel Architecture

Programming, Performance, & Efficiency

Rama Malladi

Intel, Bangalore

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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.

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.

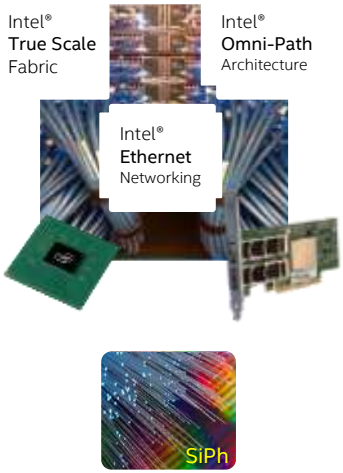
Notice revision #20110804

Intel Technical Computing & HPC
Solutions Portfolio

Compute Processing

Systems & Boards

Network & Fabric



I/O & Storage

Software & Services

Growth Trends

- Core and Thread Count
- Dual precision (DP) Flops
- Power efficiency
- Memory Bandwidth
- Transistor Scaling

Multi-Core

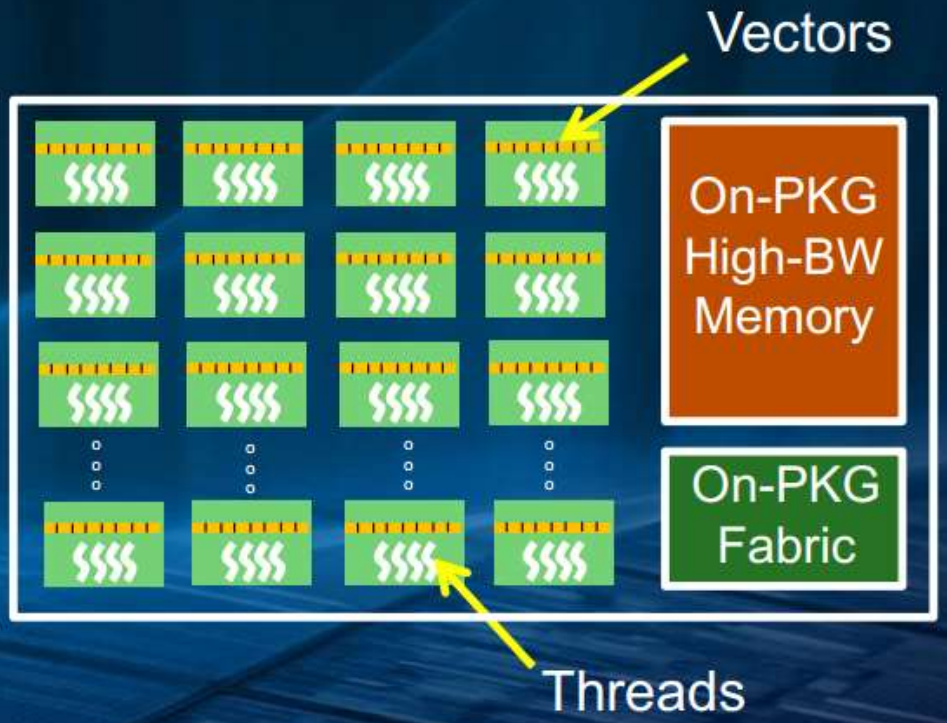
Optimized for **Serial and Parallel** Apps

Many-Core

Optimized for **Highly Parallel and Highly Vectorized** Apps

High Performance Processor Trend

- Many IA Cores
- Lots of IA Threads
- Lots of Wide Vectors
- Coherent Cache Hierarchy
- Large On-PKG high-bandwidth Memory in addition to DDR
- On-PKG Fabric
- Standalone general purpose CPU



Agenda

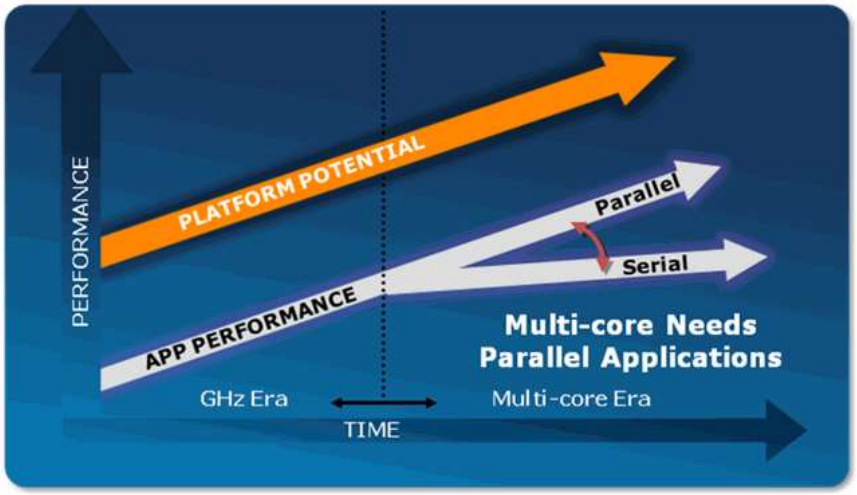
- The problem statement!
- Intel® Xeon® Processor Architecture Basics
- Intel® Xeon Phi™ (Knights Landing)
- Software Tools for Developers
- Summary

Parallel & Vector Execution

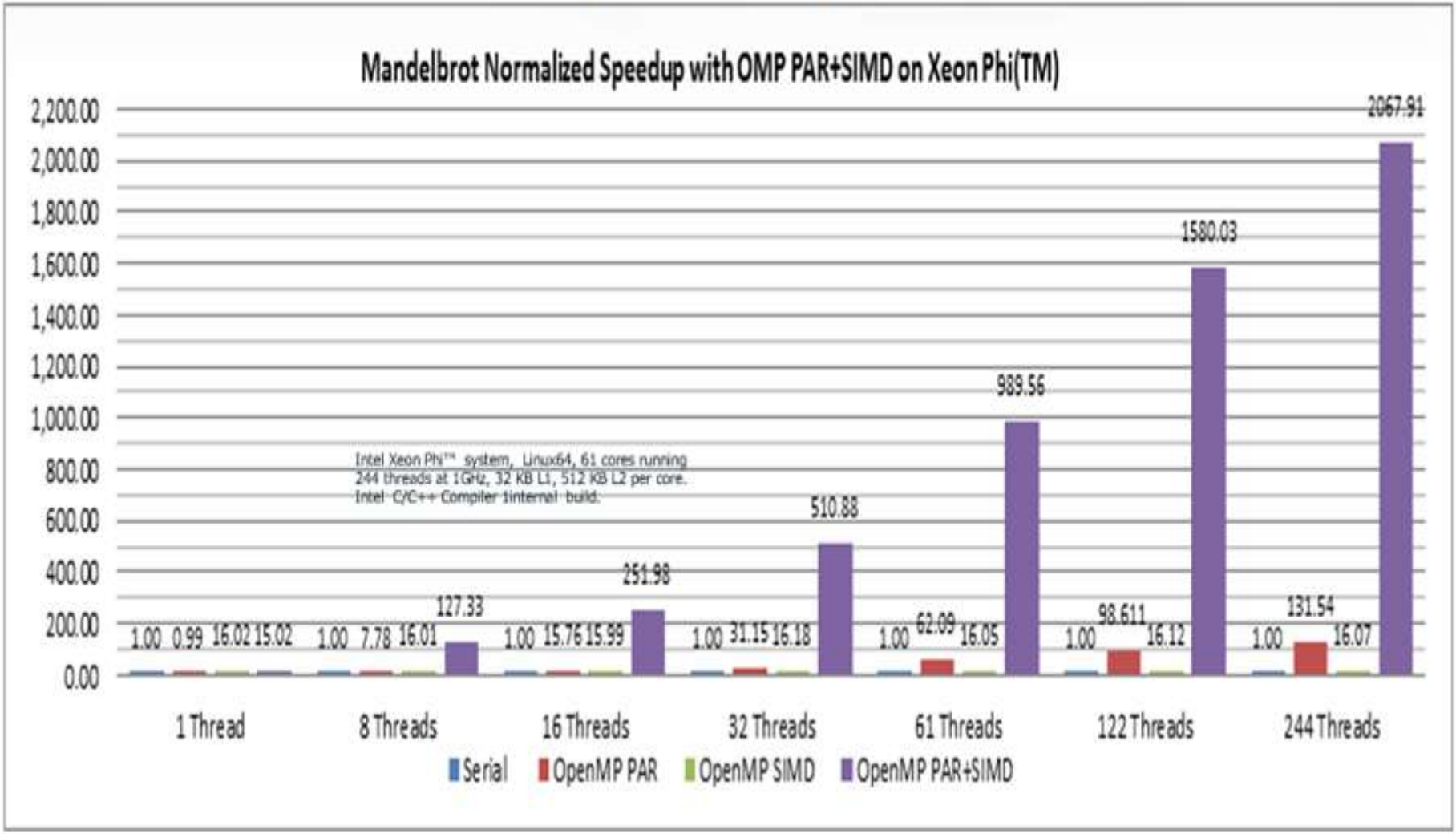
Problem:
Economical operation **frequency of (CMOS) transistors is limited**.
⇨ **No free lunch anymore!**

Solution:
More transistors allow more gates/logic on the same die space and power envelop, **improving parallelism**:

- **Thread level** parallelism (TLP):
  Multi- and many-core
- **Data level** parallelism (DLP):
  Wider vectors (SIMD)
- **Instruction level** parallelism (ILP):
  Microarchitecture improvements, e.g. threading, superscalarity, …



Mandelbrot: Speedup on Xeon Phi™



Optimized & Modernized Implementation

recap

- ✓ Loop Unrolling (#pragma unroll)
  - Short loop hurts instruction scheduling.
- ✓ Threading (#pragma omp parallel)
  - Embarrassingly parallel.
  - No write conflicts and small working set.
- ✓ Vectorization (#pragma omp simd)
  - v0/v1 must be reduced.
  - max() call introduces control divergence.
  - m_r[p] should be aligned.
- ✓ Arithmetic
  - Use native exp2() call on coprocessor.

```
#pragma omp target device(0)
#pragma omp parallel for
for(int o = 0; o < nopt; o++)
{
  const REAL_T _rt_tLN2=sqrt(T[o])*vol/M_LN2;
  const REAL_T mu_tLN2 = T[o]*mu/MLN2;
  REAL_T v0 = 0, v1 = 0, res;
  #pragma omp simd  reduction(+:v0,v1)
  aligned(m_r:64) unroll(4)
  for(int p = 0; p < npath; ++p) {
    res = max(0, S[o]*exp2(v_rt_tLN2*m_r[p]
                + mu_tLN2)-X[o]);
    v0 += res;
    v1 += res*res;
  }
  result    [o] += v0;
  confidence[o] += v1;
}
```

Roofline Analysis?

How to determine if we got the best / peak performance?

- Run GEMM?
- LINPACK?
- STREAM bandwidth tests?
- Latency benchmarks?
- …
- Get theoretical peak possible for the code?

Roofline Analysis

“paper-pen” exercise

```
float *A, *B, *C, d;
for(i=0; i<n; i++)
{
  A[i] = B[i] + d * C[i];
}
```

The above code on Intel Xeon Phi is bound by:

- Compute?
- Bandwidth?

Agenda

- The problem statement!
- Intel® Xeon® Processor Architecture Basics
- Intel® Xeon Phi™ (Knights Landing)
- Software Tools for Developers
- Summary

History

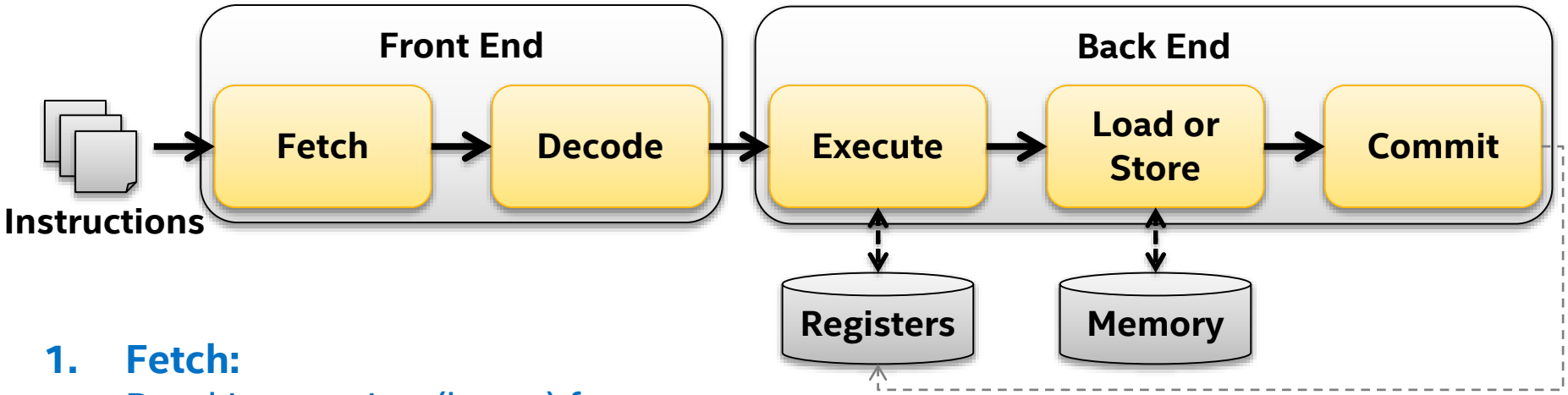
Intel® 64 and IA-32 Architecture

- 1978: 8086 and 8088 processors
- 1982: Intel® 286 processor
- 1985: Intel386™ processor
- 1989: Intel486™ processor
- 1993: Intel® Pentium®
- 1995-1999: P6 family:
  - Intel Pentium Pro processor
  - Intel Pentium II [Xeon] processor
  - Intel Pentium III [Xeon] processor
  - Intel Celeron® processor
- 2000-2006: Intel® Pentium® 4 processor
- 2005: Intel® Pentium® processor Extreme Edition
- 2001-2007: Intel® Xeon® processor
- 2003-2006: Intel® Pentium® M processor
- 2006/2007: Intel® Core™ Duo and Intel® Core™ Solo processors
- 2006/2007: Intel® Core™2 processor
- 2008: Intel® Atom™ processor and Intel® Core™ i7 processor family
- **2010: Intel® Core™ processor family**
- **2011: Second generation Intel® Core™ processor family**
- **2012: Third generation Intel® Core™ processor family**
- **2013: Fourth generation Intel® Core™ processor family**
- **2013: Intel® Atom™ processor family based on Silvermont microarchitecture**

Processor Architecture Basics

Pipeline

Computation of instructions requires several stages:



1. **Fetch:**
   Read instruction (bytes) from memory
2. **Decode:**
   Translate instruction (bytes) to microarchitecture
3. **Execute:**
   Perform the operation with a functional unit
4. **Memory:**
   Load (read) or store (write) data, if required
5. **Commit:**
   Retire instruction and update micro-architectural state

Processor Architecture Basics

Reducing Impact of Pipeline Stalls

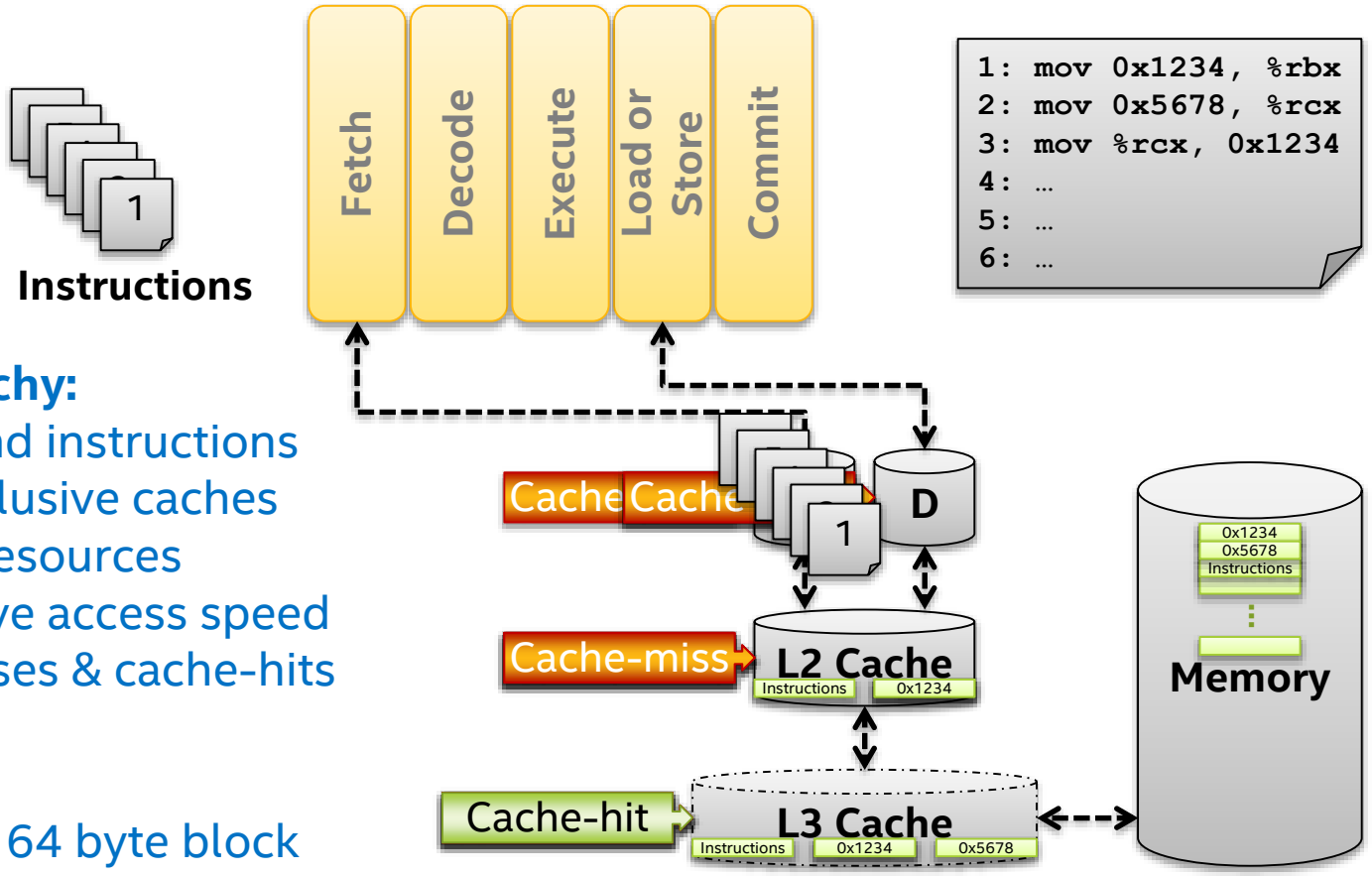
Impact of pipeline stalls can be reduced by:

- Branch prediction
- Superscalarity + multiple issue fetch & decode
- Out of Order execution
- Cache
- Non-temporal stores
- Prefetching
- Line fill buffers
- Load/Store buffers
- Alignment
- Simultaneous Multithreading (SMT)

⇨ **Characteristics of the architecture that might require user action!**

Processor Architecture Basics

Cache-Hierarchy & Cache-Line



Cache-hierarchy:

- For data and instructions
- Usually inclusive caches
- Races for resources
- Can improve access speed
- Cache-misses & cache-hits

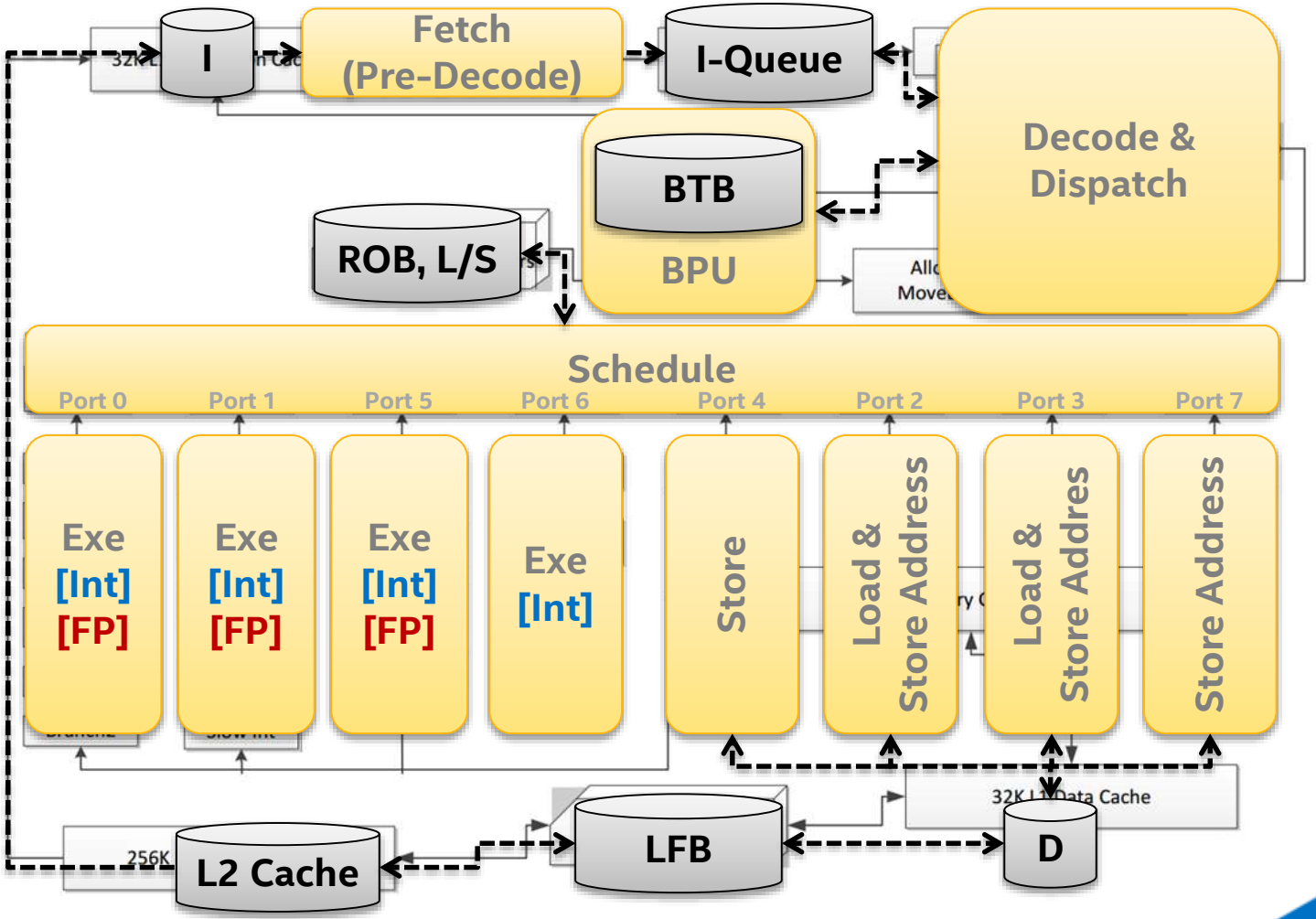
Cache-line:

- Always full 64 byte block
- Minimal granularity of every load/store
- Modifications invalidate entire cache-line (dirty bit)

Processor Architecture Basics

Example: 4th Generation Intel® Core™

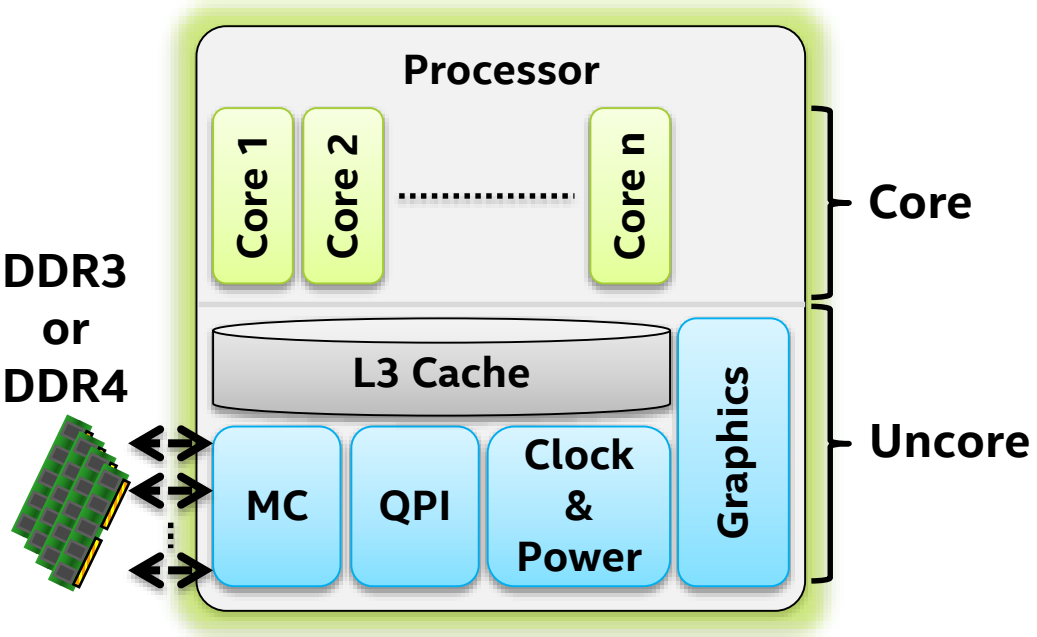
From Intel® 64 and IA-32 Architectures Optimization Reference Manual:



Processor Architecture Basics

Core vs. Uncore

- **Core:**
  Processor core's logic:
  - Execution units
  - Core caches (L1/L2)
  - Buffers & registers
  - …



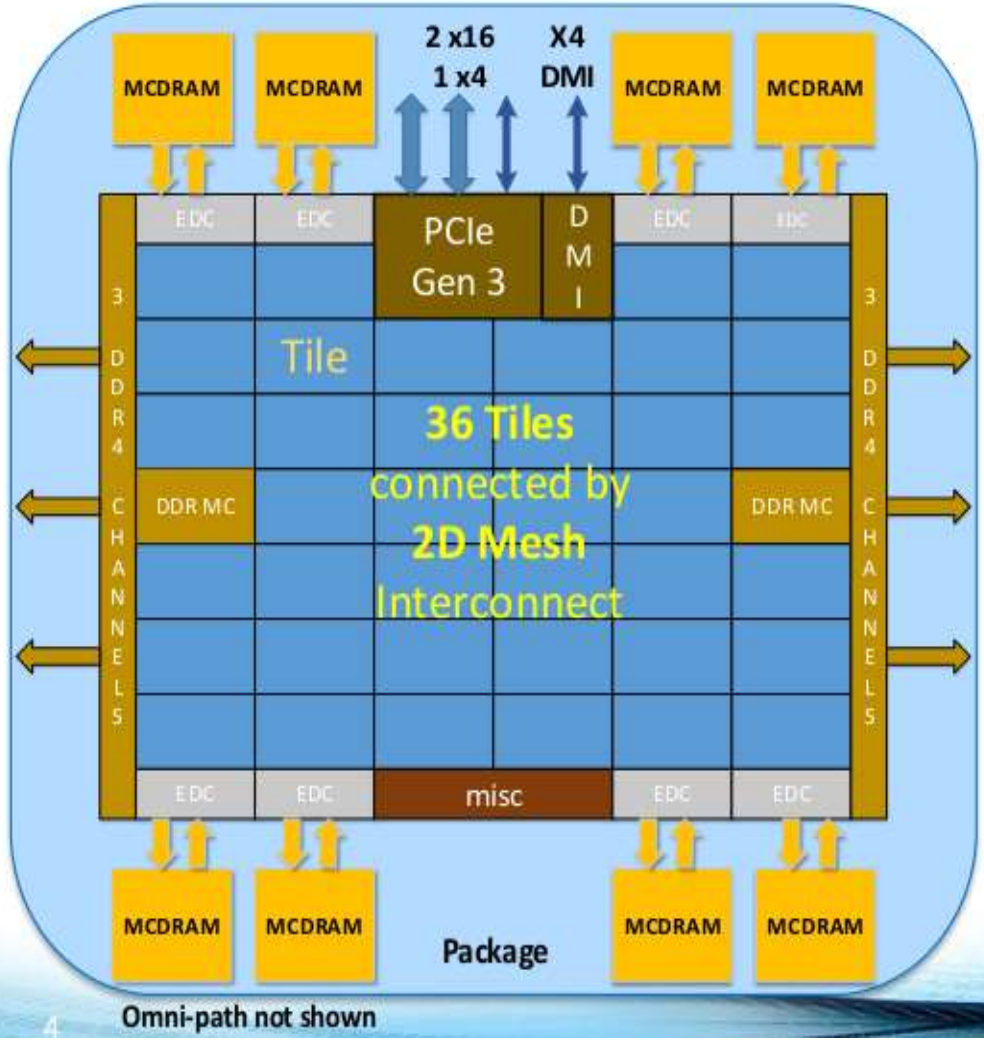
- **Uncore:**
  All outside a processor core:
  - Memory controller/channels (MC) and Intel® QuickPath Interconnect (QPI)
  - L3 cache shared by all cores
  - Type of memory
  - Power management and clocking
  - Optionally: Integrated graphics

⇨ **Only uncore is differentiation within same processor family!**

Agenda

- The problem statement!
- Intel® Xeon® Processor Architecture Basics
- Intel® Xeon Phi™ (Knights Landing)
- Software Tools for Developers
- Summary

Knights Landing Overview



Chip: 36 Tiles interconnected by **2D Mesh**

Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW

DDR4: 6 channels @ 2400 up to 384GB

IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset

Node: 1-Socket only

Fabric: Omni-Path on-package (not shown)

Vector Peak Perf: 3+TF DP and 6+TF SP Flops

Scalar Perf: ~3x over Knights Corner

Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set Architecture (ISA). *Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as flat memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system [illegible] actual performance.

Knights Landing: Next Intel® Xeon Phi™ Processor

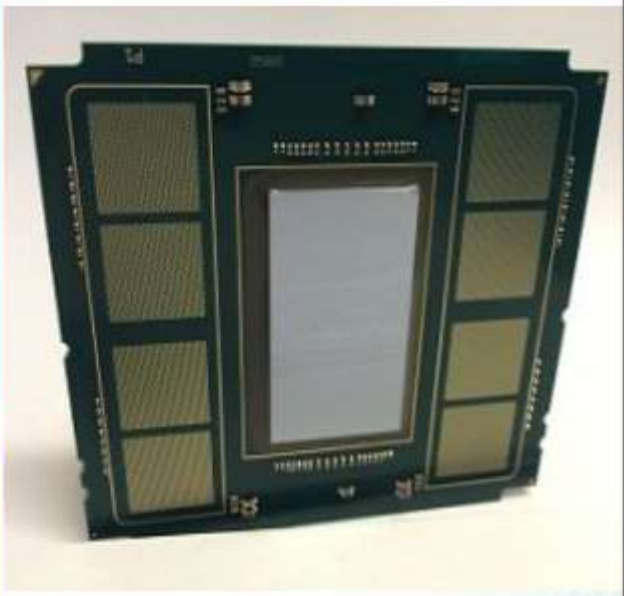
Intel® Many-Core Processor targeted for HPC and Supercomputing

First **self-boot** Intel® Xeon Phi™ processor that is **binary compatible** with main line IA. Boots standard OS.

Significant improvement in scalar and **vector** performance

Integration of **Memory on package**: innovative memory architecture for high bandwidth and high capacity

Integration of **Fabric on package**



Three products

KNL Self-Boot	KNL Self-Boot w/ Fabric	KNL Card
(Baseline)	(Fabric Integrated)	(PCIe-Card)

Potential future options subject to change without notice.
All timeframes, features, products and dates are preliminary forecasts and subject to change without further notification.

Many Trailblazing Improvements in KNL

Improvements	What/Why
Self Boot Processor	No PCIe bottleneck
Binary Compatibility with Xeon	Runs all legacy software. No recompilation.
New Core: Atom™ based	~3x higher ST performance over KNC
Improved Vector density	3+ TFLOPS (DP) peak per chip
New AVX 512 ISA	New 512-bit Vector ISA with Masks
Scatter/Gather Engine	Hardware support for gather and scatter
New memory technology: MCDRAM + DDR	Large High Bandwidth Memory → MCDRAM Huge bulk memory → DDR
New on-die interconnect: Mesh	High BW connection between cores and memory
Integrated Fabric: Omni-Path	Better scalability to large systems. Lower Cost

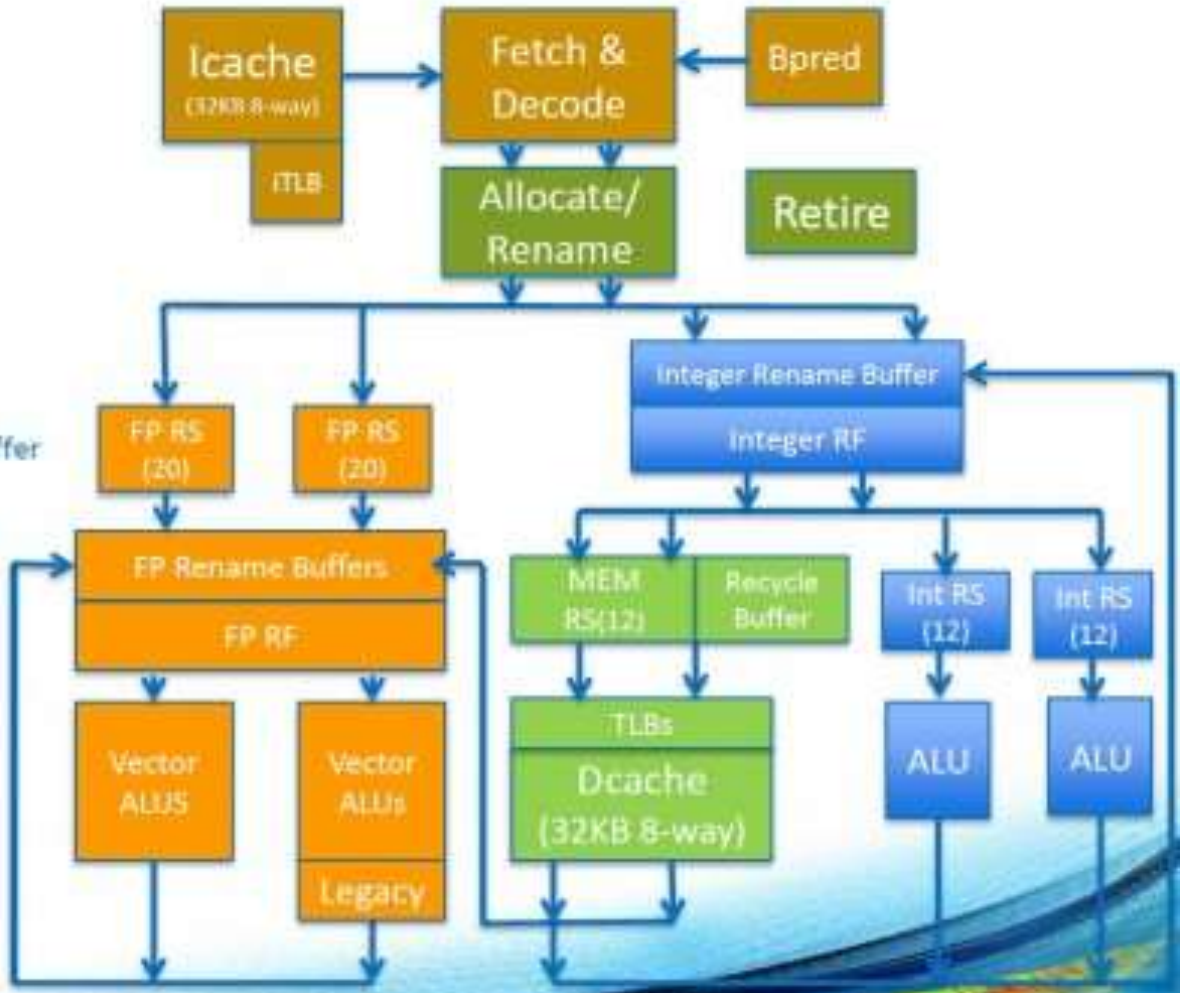
Core & VPU

- Out-of-order core w/ 4 SMT threads
- VPU tightly integrated with core pipeline

- 2-wide Decode/Rename/Retire
- ROB-based renaming. 72-entry ROB & Rename Buffers
- Up to 6-wide at execution
- Int and FP RS OoO.
- MEM RS inorder with OoO completion. Recycle Buffer holds memory ops waiting for completion.
- Int and Mem RS hold source data. FP RS does not.

- 2x 64B Load & 1 64B Store ports in Dcache.
- 1st level uTLB: 64 entries
- 2nd level dTLB: 256 4K, 128 2M, 16 1G pages

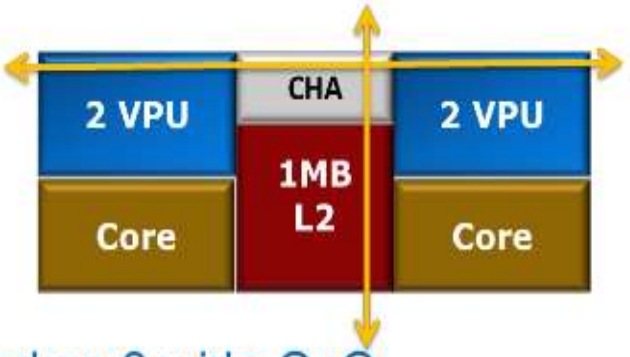
- L1 Prefetcher (IPP) and L2 Prefetcher.
- 46/48 PA/VA bits
- Fast unaligned and cache-line split support.
- Fast Gather/Scatter support



KNL Tile:

2 Cores, each with 2 VPU

1M L2 shared between two Cores



Core: Changed from Knights Corner (KNC) to KNL. Based on 2-wide OoO Silvermont™ Microarchitecture, but with *many* changes for HPC.

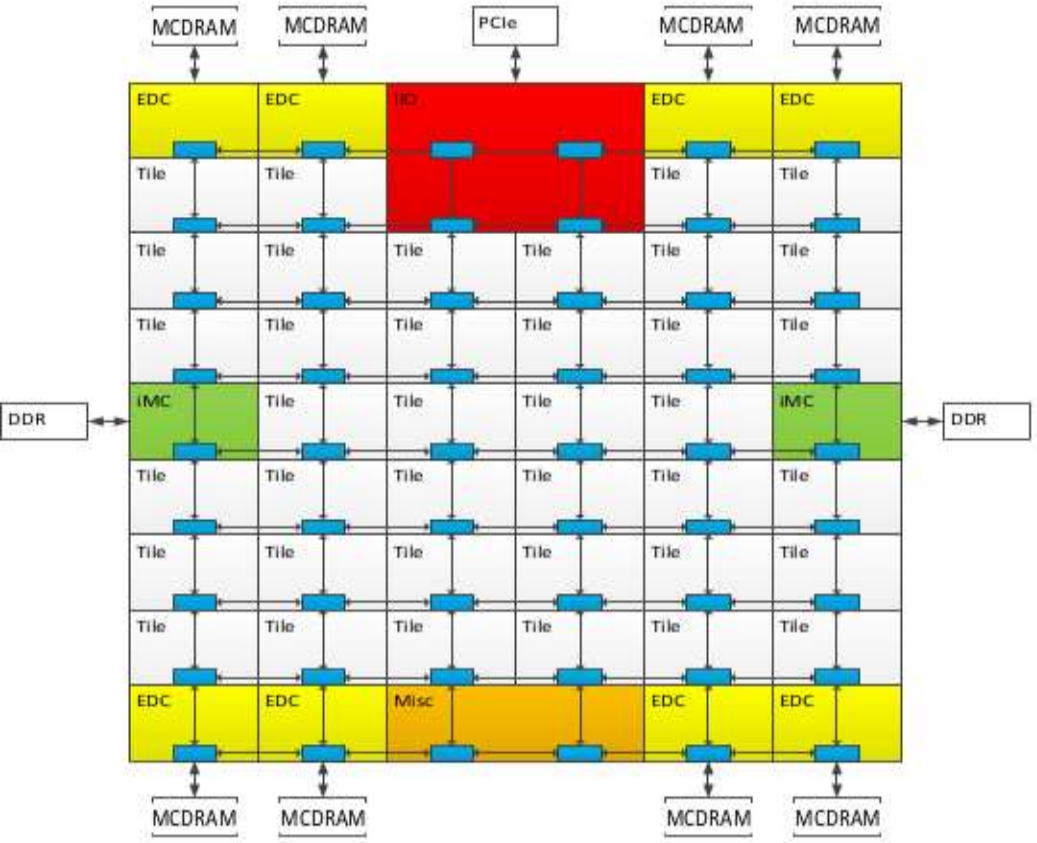
4 thread/core. Deeper OoO. Better RAS. Higher bandwidth. Larger TLBs.

2 VPU: 2x AVX512 units. 32SP/16DP per unit. X87, SSE, AVX1, AVX2 and EMU

L2: 1MB 16-way. 1 Line Read and ½ Line Write per cycle. Coherent across all Tiles

CHA: **C**aching/**H**ome **A**gent. Distributed Tag Directory to keep L2s coherent. MESIF protocol. 2D-Mesh connections for Tile

KNL Mesh Interconnect



Mesh of Rings

- Every row and column is a (half) ring
- YX routing: Go in Y → Turn → Go in X
- Messages arbitrate at injection and on turn

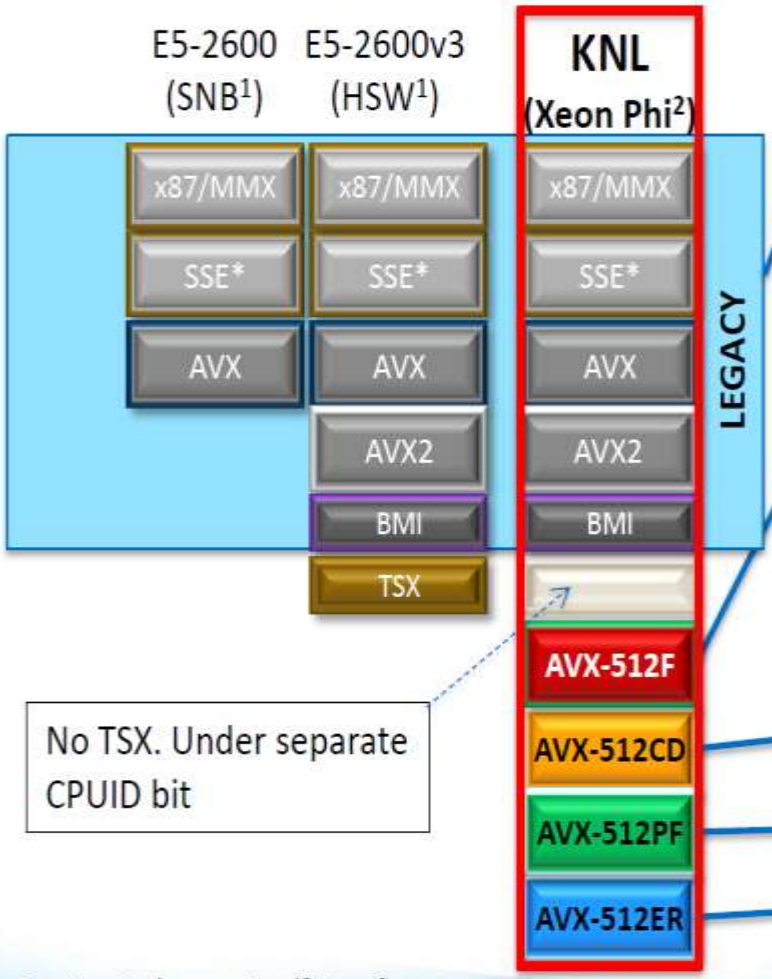
Cache Coherent Interconnect

- MESIF protocol (F = Forward)
- Distributed directory to filter snoops

Three Cluster Modes

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

KNL ISA



No TSX. Under separate CPUID bit

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

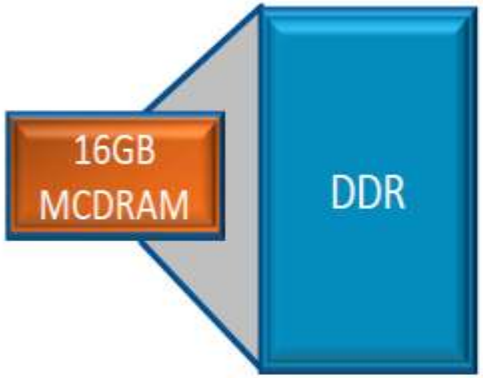
Exponential and Reciprocal Instructions

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

Memory Modes

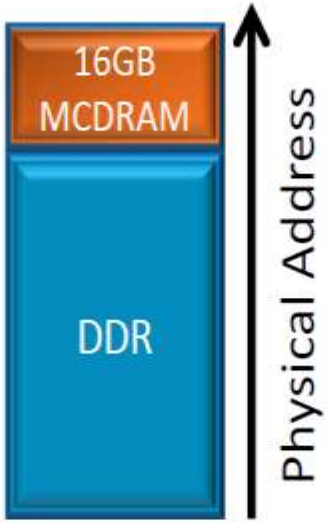
Three Modes. Selected at boot

Cache Mode



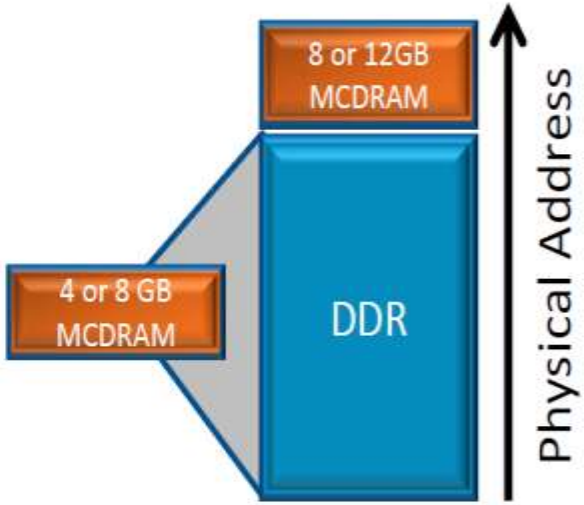
- SW-Transparent, Mem-side cache
- Direct mapped. 64B lines.
- Tags part of line
- Covers whole DDR range

Flat Mode



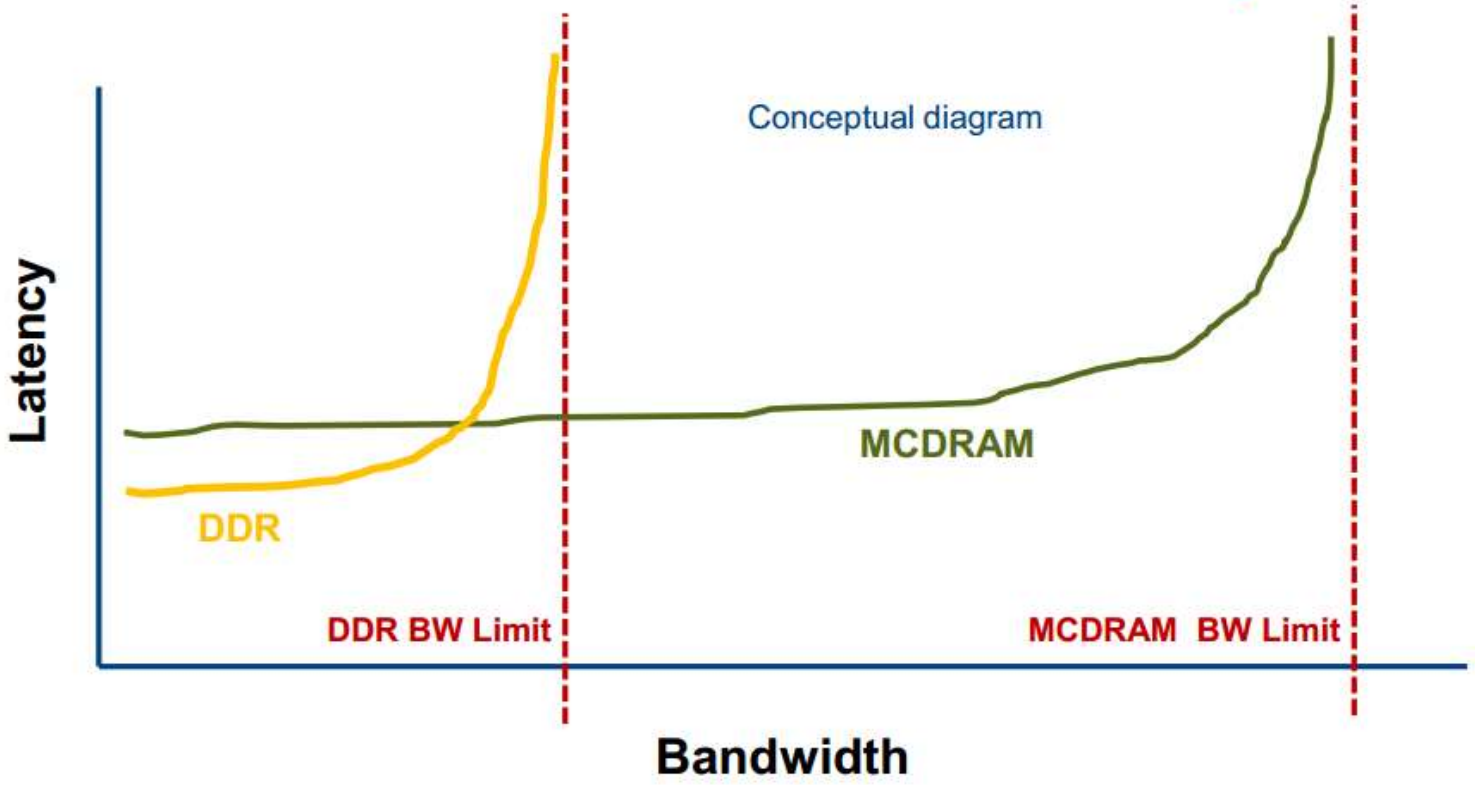
- MCDRAM as regular memory
- SW-Managed
- Same address space

Hybrid Mode



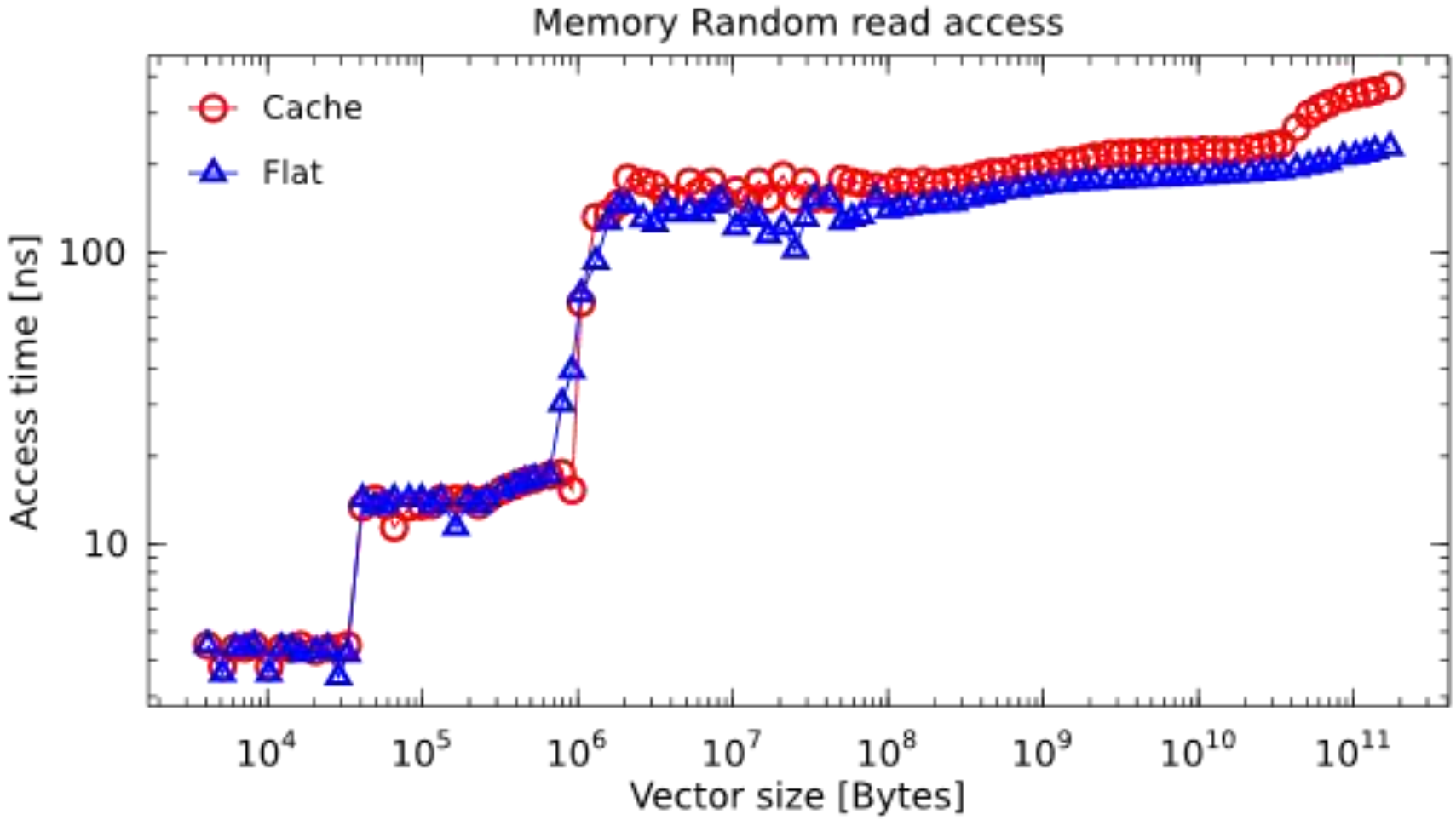
- Part cache, Part memory
- 25% or 50% cache
- Benefits of both

DDR and MCDRAM Bandwidth vs. Latency



MCDRAM latency more than DDR at low loads but much less at high loads

Random Access Latency



Flat MCDRAM SW Usage: Code Snippets

C/C++ (*https://github.com/memkind)

Allocate into DDR

```
float   *fv;
fv = (float *)malloc(sizeof(float)*100);
```

Allocate into MCDRAM

```
float   *fv;
fv = (float *)hbw_malloc(sizeof(float) * 100);
```

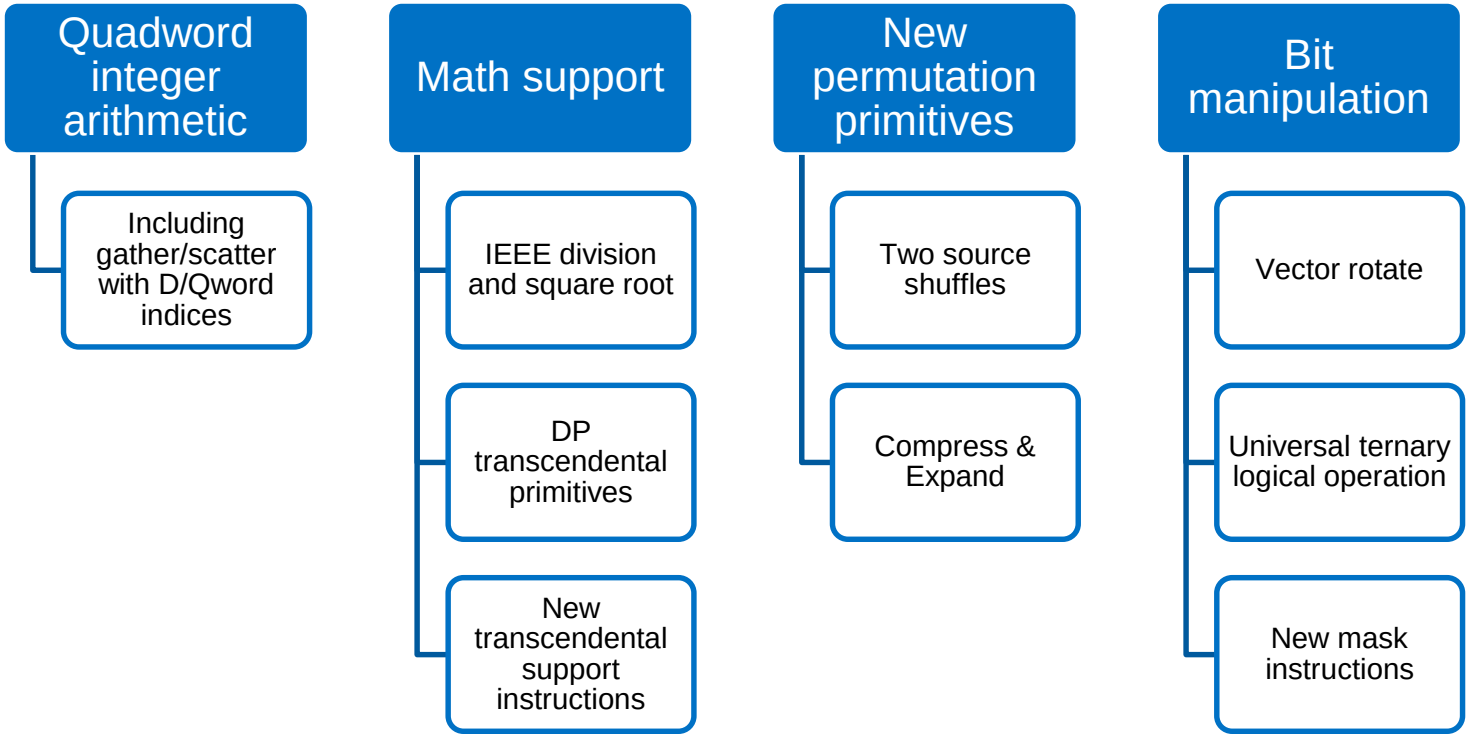
Intel Fortran

Allocate into MCDRAM

```
c      Declare arrays to be dynamic
       REAL, ALLOCATABLE :: A(:)

!DEC$ ATTRIBUTES, FASTMEM :: A

       NSIZE=1024
c      allocate array 'A' from MCDRAM
c
       ALLOCATE (A(1:NSIZE))
```

AVX-512 Designed for HPC

- Promotions of many AVX and AVX2 instructions to AVX-512
  - 32-bit and 64-bit floating-point instructions from AVX
    - Scalar and 512-bit
  - 32-bit and 64-bit integer instructions from AVX2
- Many new instructions to speedup HPC workloads



AVX-512 features (I): More & Bigger Registers

AVX: VADDPS YMM0, YMM3, [mem]

- Up to 16 AVX registers
  - 8 in 32-bit mode
- 256-bit width
  - 8 x FP32
  - 4 x FP64

```
float32 A[N], B[N];

for(i=0; i<8; i++)
{
    A[i] = A[i] + B[i];
}
```

AVX-512: VADDPS ZMM0, ZMM24, [mem]

- Up to 32 AVX registers
  - 8 in 32-bit mode
- 512-bit width
  - 16 x FP32
  - 8 x FP64

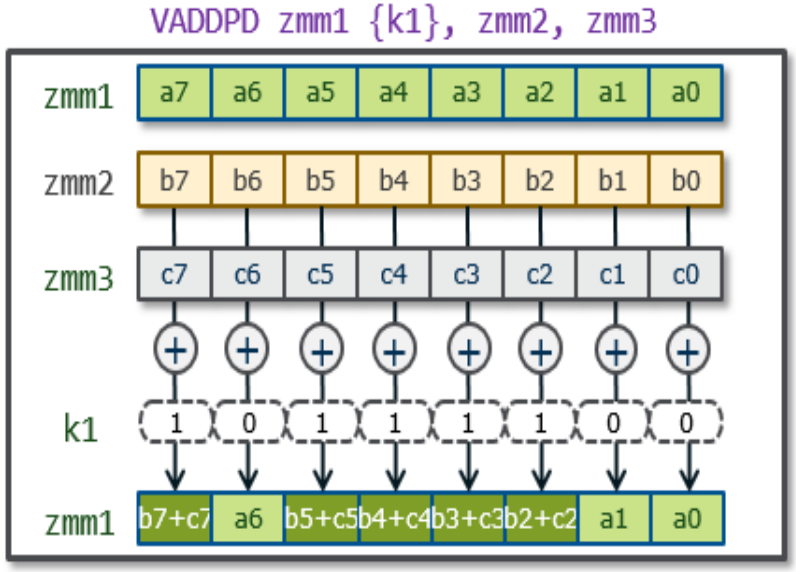
```
float32 A[N], B[N];

for(i=0; i<16; i++)
{
    A[i] = A[i] + B[i];
}
```

But you need many more features
to use all that real estate effectively…

AVX-512 Mask Registers

8 Mask registers of size 64-bits

- k1-k7 can be used for predication
  - k0 can be used as a destination or source for mask manipulation operations
  - k0 cannot be used as input mask for vector operations
    - k0 encoding treated as "no mask"



4 different mask granularities.
For instance, at 512b:

- Packed Integer Byte use mask bits [63:0]
  - `VPADDB zmm1 {k1}, zmm2, zmm3`
- Packed Integer Word use mask bits [31:0]
  - `VPADDW zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP32 and Integer Dword use mask bits [15:0]
  - `VADDPS zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP64 and Integer Qword use mask bits [7:0]
  - `VADDPD zmm1 {k1}, zmm2, zmm3`

		Vector Length		
		128	256	512
element size	Byte	16	32	64
	Word	8	16	32
	Dw ord/SP	4	8	16
	Qw ord/DP	2	4	8

Gather & Scatter

D/Q/SP/DP element types
D/Q indices
Instruction can partially execute
k-reg Mask used as completion mask

```
for(j=0, i=0; i<N; i++)
{
        B[R[i]] = A[Q[i]];
}
```

VMOVDQU64 zmm1, Q[rsi]

VMOVDQU64 zmm2, R[rsi]

VGATHERQQ zmm0 {k2}, [rax+zmm1*8]

VSCATTERQQ [rax+zmm2*8] {k3}, zmm0

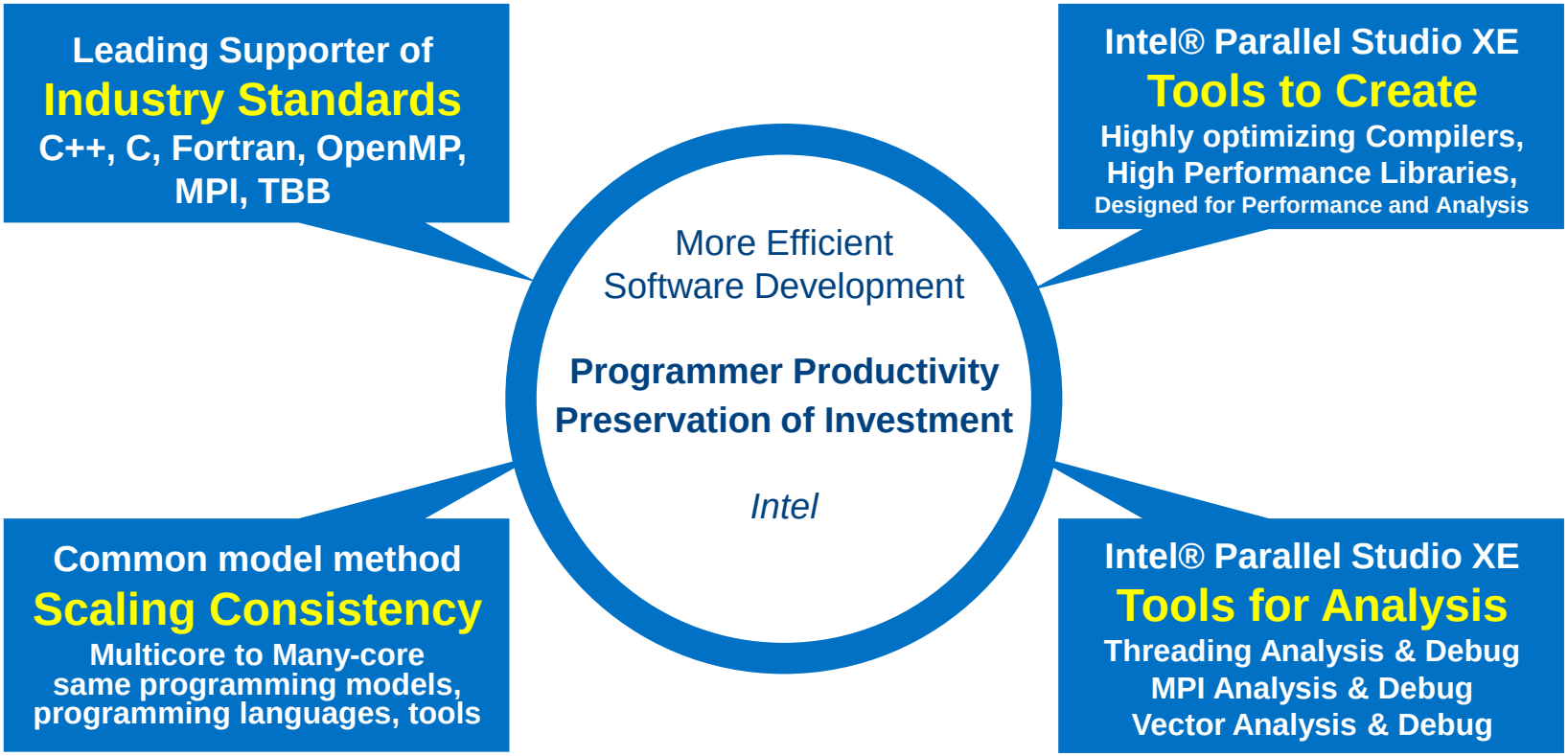
Agenda

- The problem statement!
- Intel® Xeon® Processor Architecture Basics
- Intel® Xeon Phi™ (Knights Landing)
- Software Tools for Developers
- Summary

Petascale Today

Intel® Cluster Studio XE

Intel® Cluster Studio XE 2018



Compiler Support

Intel Compilers

Compiler	Suggested flags
Intel C compiler	-O3 -xMIC-AVX512 -fma -align -finline-functions
Intel C++ compiler	-std=c11 -O3 -xMIC-AVX512 -fma -align -finline-functions
Intel Fortran compiler	-O3 -xMIC-AVX512 -fma -align array64byte -finline-functions

GNU Compilers

Default flag	Description
-O0	Reduce compilation time and make debugging produce the expected results.
-fno-inline	Do not expand any functions inline apart from those marked with the “always_inline” attribute.
-mcmodel=small	Generate code for the small code model. The program and its statically defined symbols must be within 4 GiB of each other. Pointers are 64 bits. Programs can be statically or dynamically linked.
-funderscoring	By default, GNUFortran appends an underscore to external names.
-fno-protect-parens	By default the parentheses in expression are honored for all optimization levels such that the compiler does not do any re- association. Using -fno-protect-parens allows the compiler to reorder “REAL” and “COMPLEX” expressions to produce faster code.

http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Intel Math Kernel Library

Intel MKL library

MKL Version	Link flag
Single thread, sequential	-mkl=sequential
Single thread MPI, Sequential	-mkl=cluster
Multi threaded	-mkl=parallel or -mkl

MKL contains a range of functions and other libraries. Several of the common widely used libraries and functions have been incorporated into MKL. Intel provide a large set of documentations etc about MKL, see Intel MKL documentation.

Libraries contained in MKL:

- BLAS, BLAS95 and Sparse BLAS
- FFT and FFTW (wrapper)
- LAPACK
- Direct and Iterative Sparse Solvers, including PARADISO
- ScaLAPACK with BLACS
- Vector Math
- Vector Statistics Functions

http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Small Code Example – Writing Code

A simple example of intrinsic usage is show below:

```
for(j=0; j<N; j+=8){
   __m512d vecA = _mm512_load_pd(&a[j]);
   __m512d vecB = _mm512_load_pd(&b[j]);
   __m512d vecC = _mm512_pow_pd(vecA,vecB);
   _mm512_store_pd(&c[j],vecC);
}
```

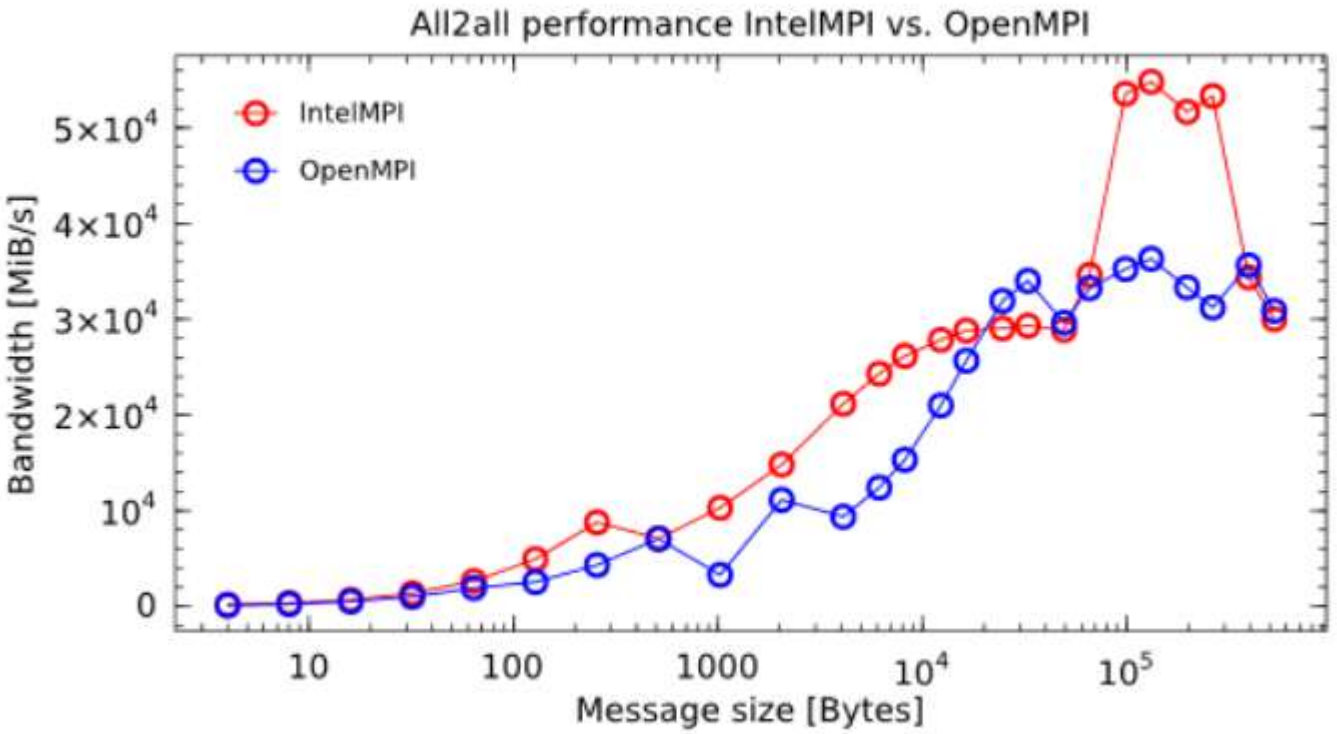
The performance can be quite good compared to the serial code below.

```
for(j=0; j<N; j++) c[j]=pow(a[j],b[j]);
```

http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

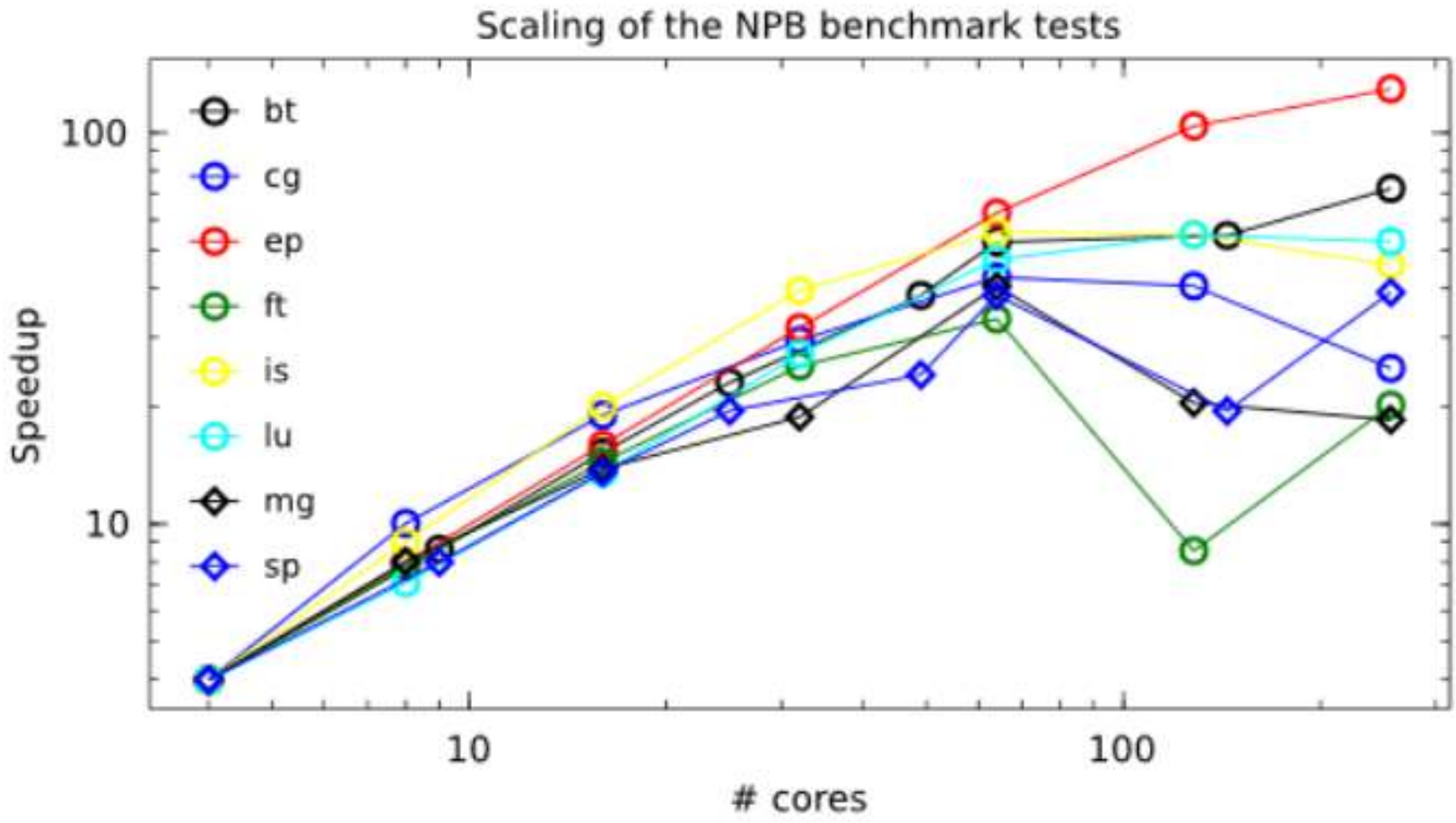
MPI Library…

MPI library	MPI CC	MPI CXX	MPI F90
Intel MPI	mpiicc	mpiicpc	mpiifort
OpenMPI	mpicc	mpicxx	mpifort



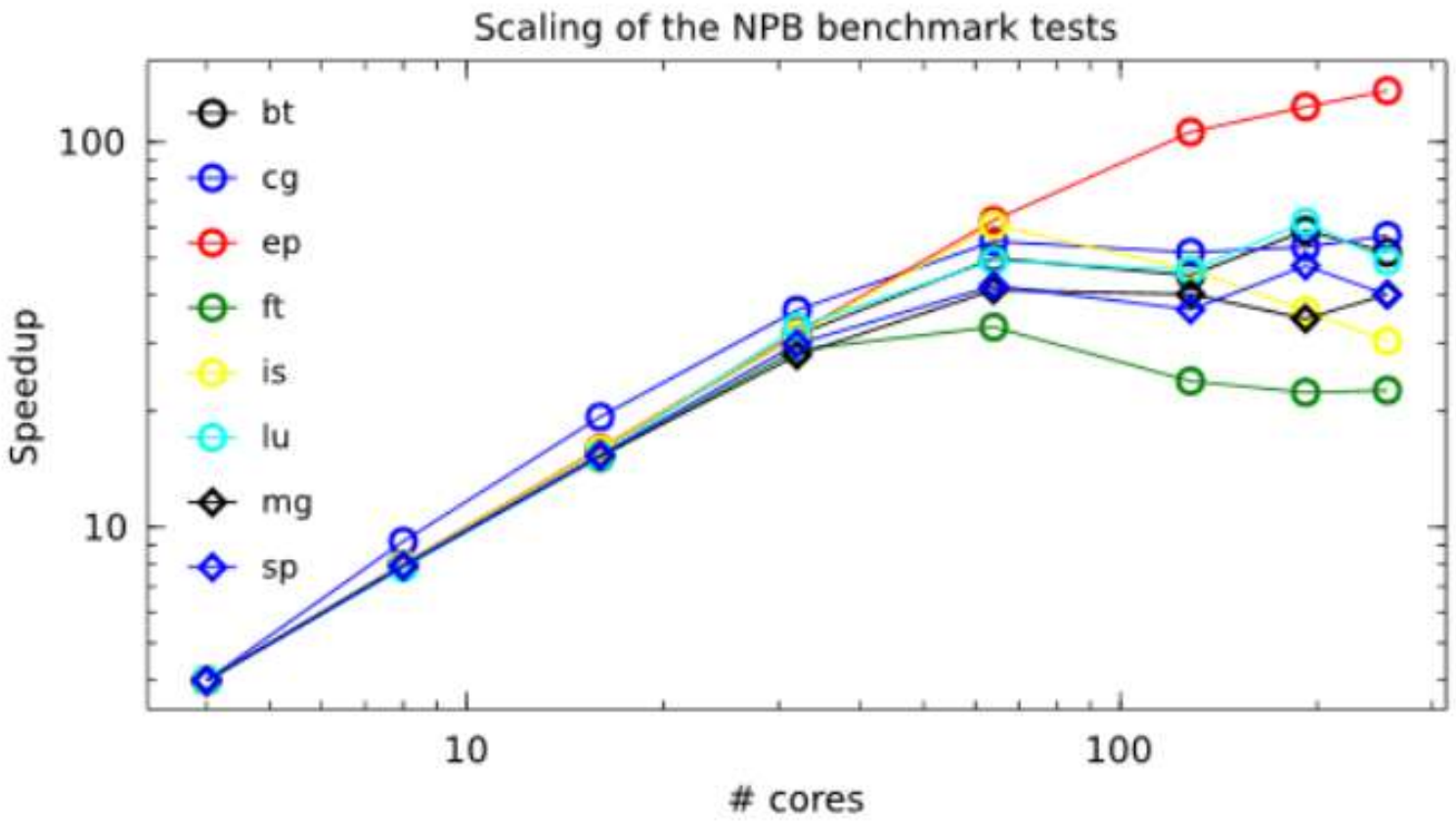
http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Scaling tests of NPB – Pure MPI



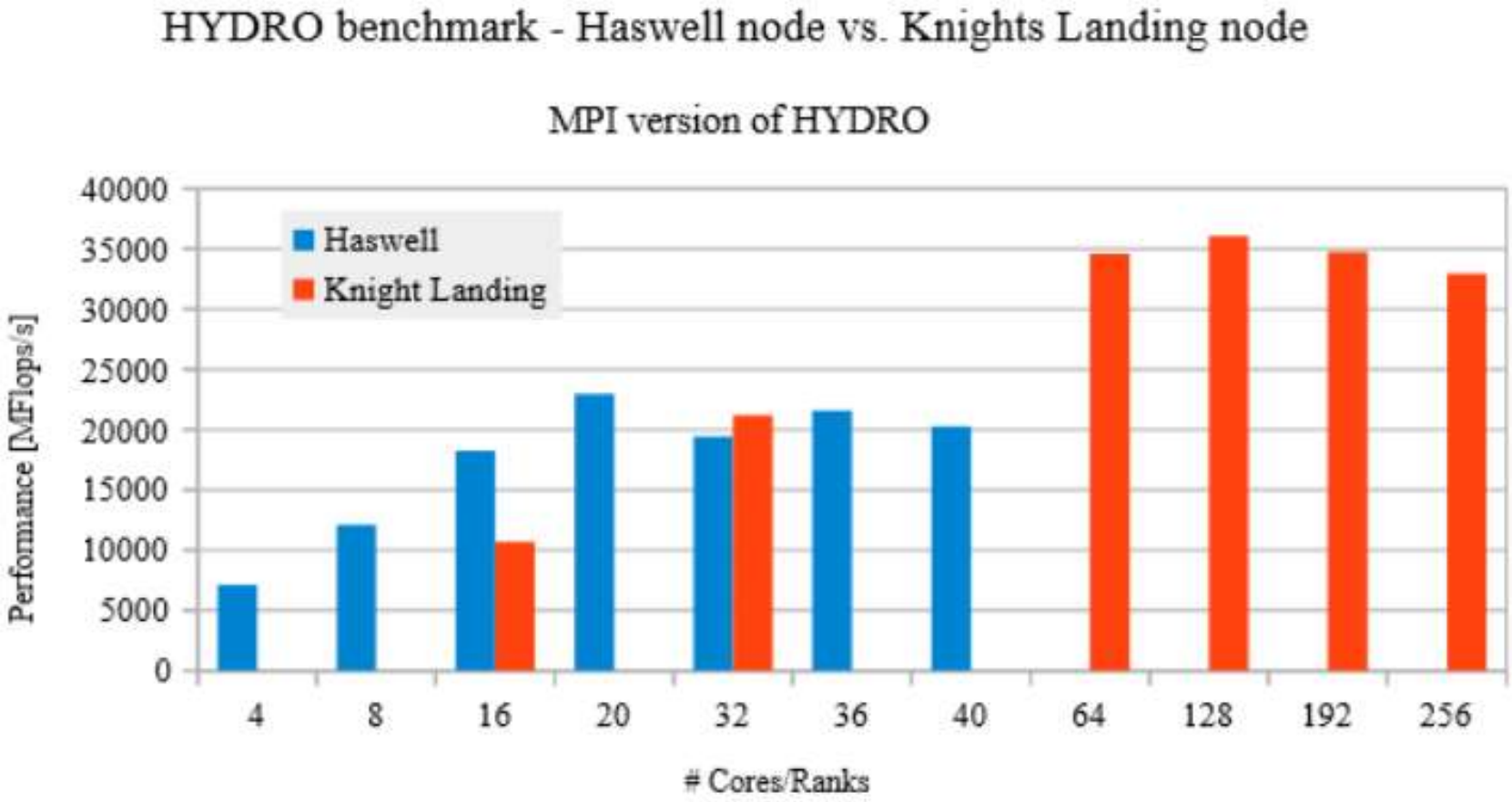
http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Scaling tests of NPB – OpenMP



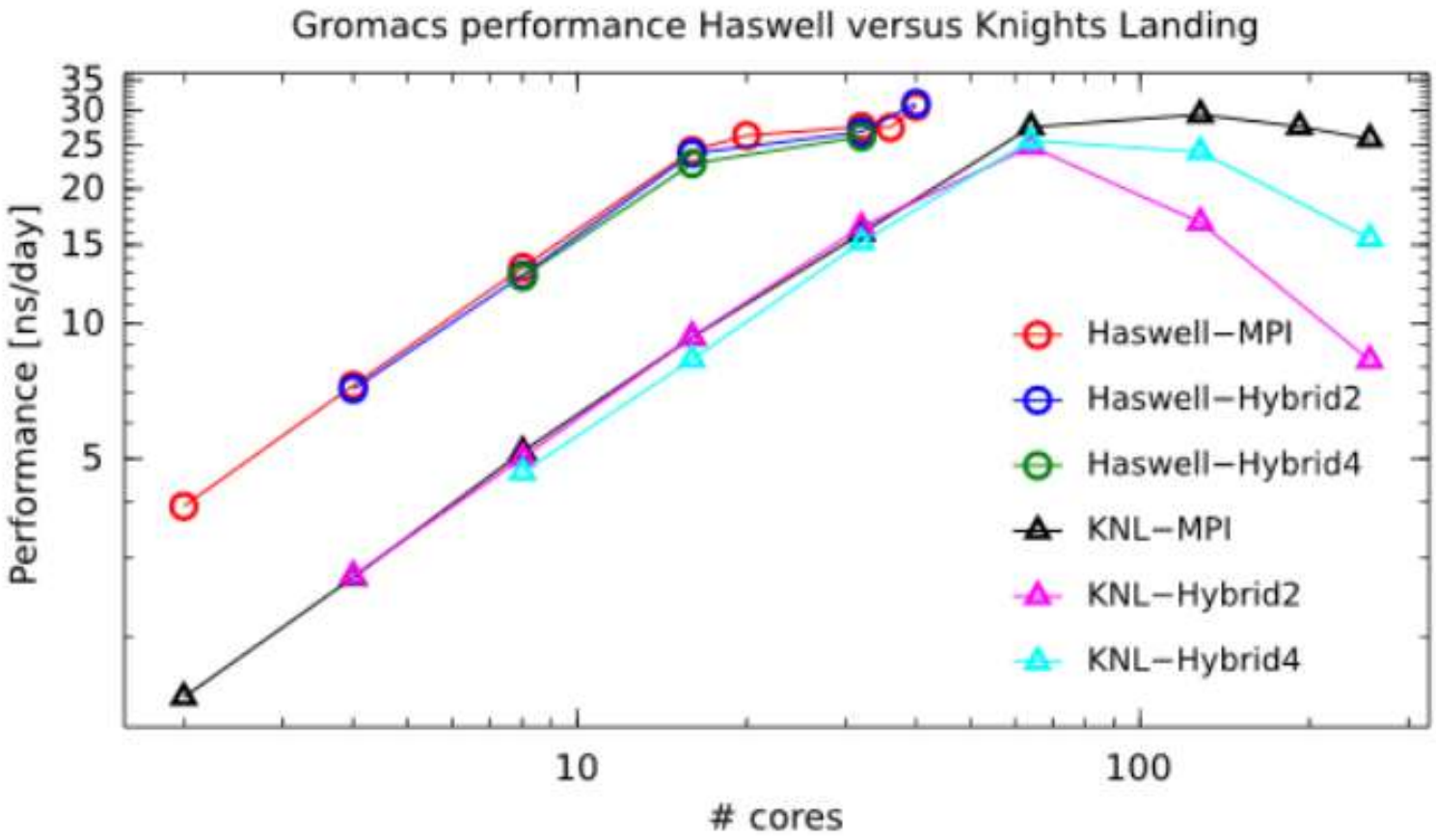
http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Scaling of HYDRO Code



http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Scaling of GROMACS



http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Compiler, Runtime Flags: GROMACS

Compiler Flags

Compiler / Library	Version
C/C++/Fortran	Intel 2017.beta (042-17.0.0-042)
MPI	Intel MPI Version 2017 Beta Build 20160302
FFTW	3.3.5 prerelease
BLAS/LAPACK	MKL (042-2017.0-042)
Compiler flags	-std=c11 -O3 -xMIC-AVX512 -align -fma -ftz -fomit-frame-pointer -finline-functions

Runtime Settings

Environment	Variable	Value
Hardware	Processor layout	Quadrant
Hardware	High Bandwidth Memory usage	Cache
MPI pinning	I_MPI_PIN	on
MPI placement	I_MPI_PIN_PROCESSOR_LIST	“processor list” (0,1,2..etc)
Thread placement	KMP_AFFINITY	“unset”
OpenMP processor binding	OMP_PROC_BIND	close

http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

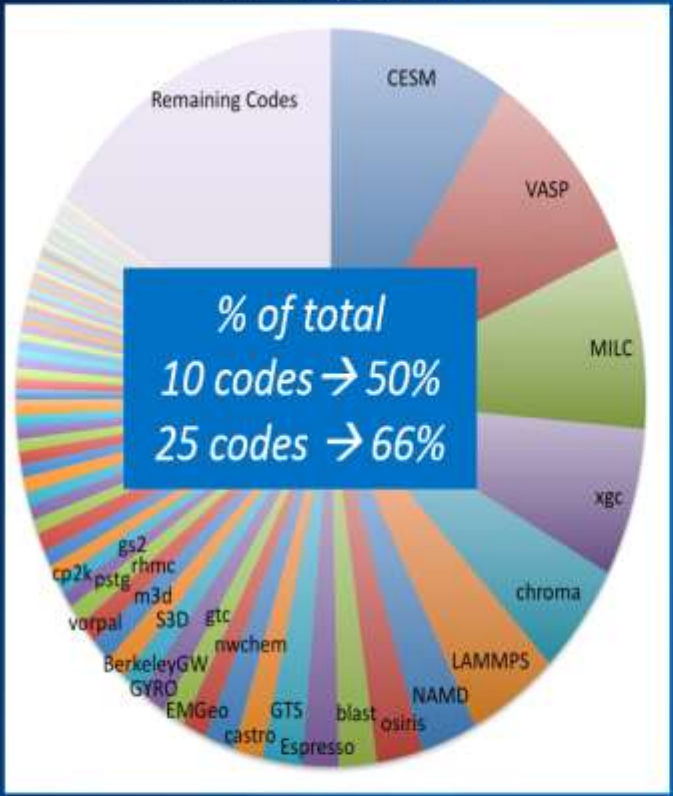
AMBER WRF VISIT VASP UTBENCH SU2 SG++ SeisSol ROTOR SIM R Quantum Espresso Optimized integral OPENMP/ MPI Openflow NWChem

Modernizing HPC Community Codes

AVBP (Large Eddy) Blast BUDE CAM-5 CASTEP CESM CFSv2 CIRCAC

NEMO5 MPAS Mardyn MACPO Ls1 Harmonie GTC GS2 Gromacs GPAW

Breakdown of Application Hours
NERSC - Hopper 2012[1]



Intel® Parallel Computing Centers

Collaborating to accelerate the pace of discovery

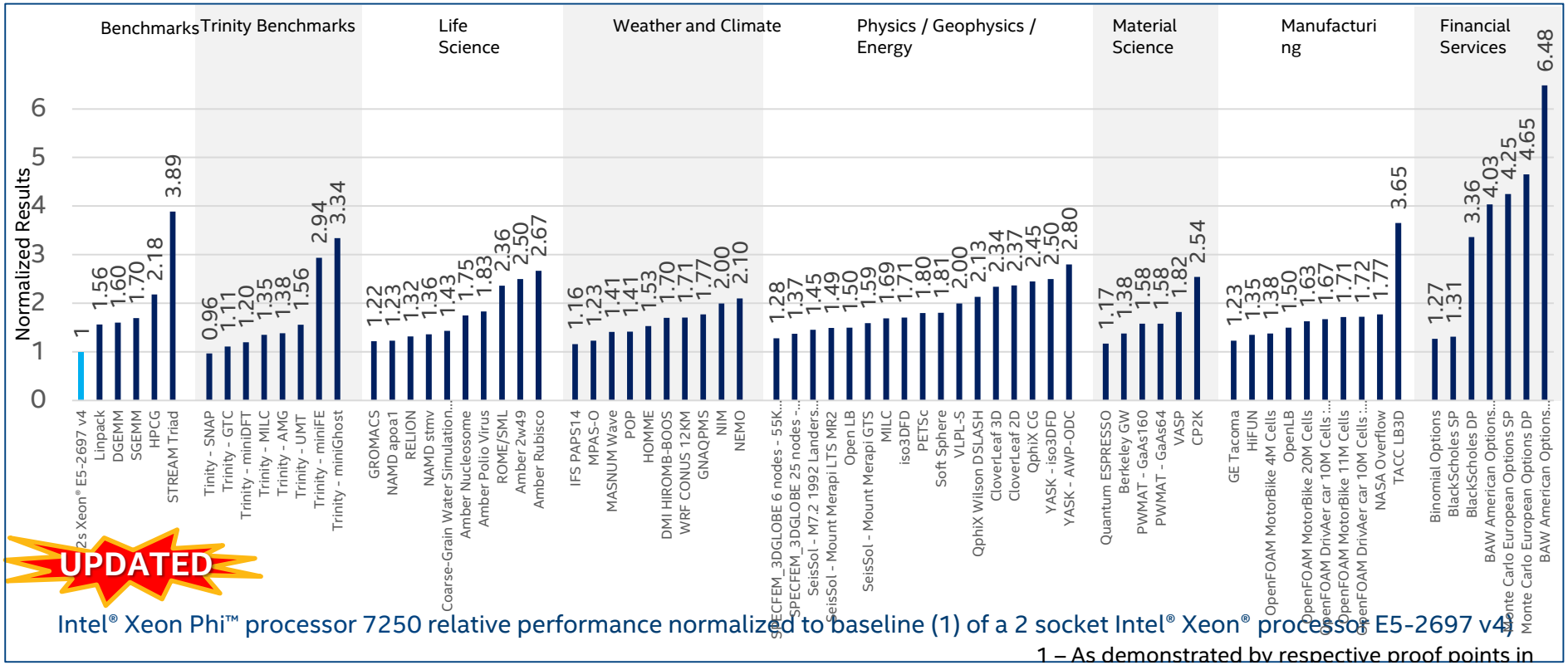
>40 Centers
13 Countries
>70 Codes
2 User Groups

https://software.intel.com/en-us/ipcc

CliPhi (COSMOS) COSA Cosmos codes DL-MESO DL-Poly ECHAM6 Elmer FrontFlow/Blue Code GADGET GAMESS-US

[1]Source: NERSC

Over 40 applications optimized for Intel® Xeon Phi™ processor family are available, with up to 6.48X (1.99X average) performance improvement[1]



UPDATED

Intel® Xeon Phi™ processor 7250 relative performance normalized to baseline (1) of a 2 socket Intel® Xeon® processor E5-2697 v4

1 – As demonstrated by respective proof points in

http://www.intel.com/performance

Summary

- Knights Landing (KNL) is the first self-boot Intel® Xeon Phi™ processor
- Many improvements for performance and programmability
  - Significant leap in scalar and vector performance
  - Significant increase in memory bandwidth and capacity
  - Binary compatible with Intel® Xeon® processor
- Common programming models between Intel® Xeon® processor and Intel® Xeon Phi™ processor
- KNL offers immense amount of parallelism (both data and thread)
  - Future trend is further increase in parallelism for both Intel® Xeon® processor and Intel® Xeon Phi™ processor
  - Developers need to prepare software to extract full benefits from this trend

Thank you!