Summer School on PARALLEL COMPUTING

Overview of the Intel Xeon and Xeon Phi tecnologies

V. Ruggiero (v.ruggiero@cineca.it)
Roma, 19 July 2017
SuperComputing Applications and Innovation Department

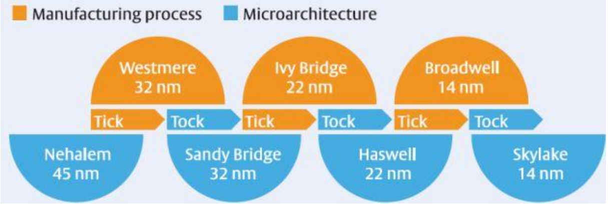
CINECA

Outline

Xeon

Xeon Phi

Tick/Tock



- Intel CPU roadmap: two step evolution
  - Tock phase:
    - New architecture
    - New instructions (ISA)
  - Tick phase:
    - Keep previous architecture
    - New technological step (e.g. Broadwell 14nm)
    - Core "optimization"
    - Typically, Increases in Transistor Density Enables New Capabilities, Higher Performance Levels, and Greater Energy Efficiency

Xeon E5-2600 v4 Product Family

- Westmere (tick, a.k.a. plx.cineca.it)
  - Intel(R) Xeon(R) CPU E5645 @2.40GHz, 6 Core per socket
- Sandy Bridge (tock, a.k.a. eurora.cineca.it)
  - Intel(R) Xeon(R) CPU E5-2687W 0 @3.10GHz, 8 core per socket
- Ivy Bridge (tick, a.k.a pico.cineca.it)
  - Intel(R) Xeon(R) CPU E5-2670 v2 @2.50GHz, 10 core per socket
- Hashwell (tock, a.k.a. galileo.cineca.it)
  - Intel(R) Xeon(R) CPU E5-2630 v3 @2.40GHz, 8 core per socket
- Broadwell (tick, a.k.a. Marconi A1)
  - Intel(R) Xeon(R) CPU E5-2699 v4 @2.3 GHz, 18 core per socket
- Slylake (tock, a.k.a. Marconi A3)
  - Intel(R) Xeon(R) CPU E5-2680v5 @2.3 GHz, 20 core per socket

Haswell vs Broadwell

Features	Xeon E5-2600 v3 (Haswell-EP)	Xeon E5-2600 v4 (Broadwell-EP)
Cores Per Socket	Up to 18	**Up to 22**
Threads Per Socket	Up to 36 threads	**Up to 44 threads**
Last-level Cache (LLC)	Up to 45 MB	**Up to 55 MB**
QPI Speed (GT/s)	2x QPI 1.1 channels 6.4, 8.0, 9.6 GT/s	
PCIe* Lanes / Speed(GT/s)	40 / 10 / PCIe* 3.0 (2.5, 5, 8 GT/s)	
Memory Population	4 channels of up to 3 RDIMMs or 3 LRDIMMs	**+ 3DS LRDIMM†**
Memory RAS	ECC, Patrol Scrubbing, Demand Scrubbing, Sparing, Mirroring, Lockstep Mode, x4/x8 SDDC	**+ DDR4 Write CRC**
Max Memory Speed	Up to 2133	**Up to 2400**
TDP (W)	160 (Workstation only), 145, 135, 120, 105, 90, 85, 65, 55	

New Comparison

Core Cache Size/Latency/Bandwidth

Metric	Nehalem	Sandy Bridge	Haswell
L1 Instruction Cache	32K, 4-way	32K, 8-way	32K, 8-way
L1 Data Cache	32K, 8-way	32K, 8-way	32K, 8-way
Fastest Load-to-use	4 cycles	4 cycles	4 cycles
Load bandwidth	16 Bytes/cycle	32 Bytes/cycle (banked)	**64 Bytes/cycle**
Store bandwidth	16 Bytes/cycle	16 Bytes/cycle	**32 Bytes/cycle**
L2 Unified Cache	256K, 8-way	256K, 8-way	256K, 8-way
Fastest load-to-use	10 cycles	11 cycles	11 cycles
Bandwidth to L1	32 Bytes/cycle	32 Bytes/cycle	**64 Bytes/cycle**
L1 Instruction TLB	4K: 128, 4-way 2M/4M: 7/thread	4K: 128, 4-way 2M/4M: 8/thread	4K: 128, 4-way 2M/4M: 8/thread
L1 Data TLB	4K: 64, 4-way 2M/4M: 32, 4-way 1G: fractured	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way
L2 Unified TLB	4K: 512, 4-way	4K: 512, 4-way	**4K+2M shared: 1024, 8-way**

All caches use 64-byte lines

15 Intel® Microarchitecture (Haswell); Intel® Microarchitecture (Sandy Bridge); Intel® Microarchitecture (Nehalem)

Broadwell Improvements

- Pure Floating-Point performances
  - Vector FP multiply latency decrease (to 3 cycles from 5)
  - Radix-1024 divider: decreased latency and increased thoughput for most divider ops.
  - Split scalar divider: Pseudo-double bandwidth for scalar divider ops
- Memory access capability
  - STLB (Software Translation Loohaside Buffer) improvements
    - Improved address prediction for branches and return
    - Provided a larger out-of-order scheduler
    - Increased size of STLB (from 1 KB to 1.5kB)

Skylake

- Improved microarchitecture
  - Improved branch predictor
  - Deeper Out-of-Order buffers
  - More execution units, shorter latencies
  - Deeper store, fill, and write-back buffers
  - Smarter prefetchers
  - Improved page miss handling
  - Better L2 cache miss bandwidth
  - Improved Hyper-Threading
  - Performance/watt enhancements
- New instructions supported
  - Memory Protection Extensions (MPX)
    - A set of processor features which, with compiler, runtime library and OS support, brings increased robustness to software by checking pointer references whose compile time normal intentions are usurped at runtime due to buffer overflow
  - AVX-512 (Xeon versions only)

Outline

Xeon

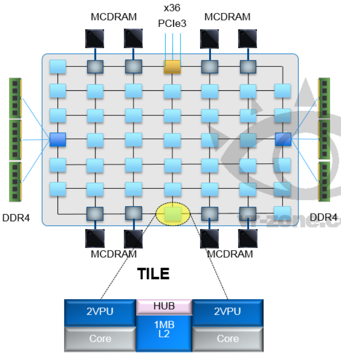
Xeon Phi

KC vs KL

Knights Corner	Knights Landing
2013	2015
22 nm	12 nm
1 TeraFLOP DP Peak	3+ TeraFLOP DP Peak
57-61 cores In-order architecure 1 Vector Unit per core	72 cores (36 tiles) Out-of-order based on Intel Atom core 2 Vector UNits per core
Intel initial Many Core instructions	Intel Advanced Vector Extension (AVX-512)

Knights Landing

Knights Landing Processor Architecture



Up to 72 Intel Architecture cores based on Silvermont (Intel® Atom processor)

- Four threads/core
- Two 512b vector units/core
- Up to 3x single thread performance improvement over KNC generation

Full Intel® Xeon processor ISA compatibility through AVX-512 (except TSX)

6 channels of DDR4 2400 MHz -up to 384GB

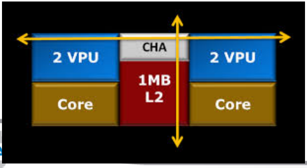
36 lanes PCI Express* Gen 3

8/16GB of high-bandwidth on-package MCDRAM memory >500GB/sec

200W TDP

KNL Core

- Core: Changed from KNC to KNL. Based on Silvermont core with many changes
  - Out of order 2-wide core: 72 inflight ops. 4 threads/core
  - Back to back fetch and issue per thread
  - 32KB Icache, 32KB Dcache. 2x 64B Loads ports in Dcache. Larger TLBs than in SLM
  - L1 Prefetcher (IPP) and L2 Prefetcher. 46/48 PA/VA bits to match Xeon
  - Fast unaligned and cache-line split support. Fast Gather/Scatter support
  - 2x BW between Dcache and L2 than in SLM: 1 line Rd and 1/2 line Wr per cycle
- 2 VPUs: 2x 512b Vectors. 32SP and 16DP.



KNL TILe: 2 Cores, each with 2 VPU, 1M L2 shared between two Cores

Many Improvements in KNL

Improvements	What/Why
Binary compatibility with Xeon	Runs all legacy software. No recompilation
New Core: SLM based	3x higher ST performance over KNC
Improved Vector density	3+ TFLOPS (DP) peak per chip
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

Core and VPU

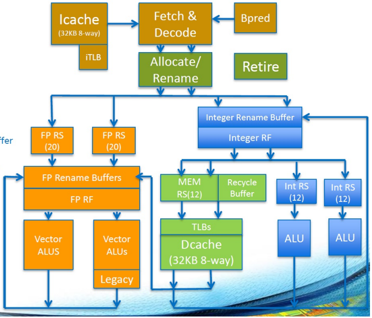
Core & VPU

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

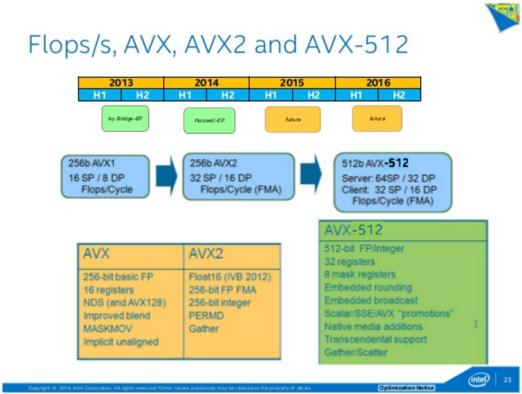
- 2-wide Decode/Rename/Retire
- ROB-based renaming. 72-entry ROB & Rename Buffers
- Up to 6-wide at execution
- Int and FP RS OoO.
- MEM RS inorder with OoO completion. Recycle Buffer holds memory ops waiting for completion.
- Int and Mem RS hold source data. FP RS does not.

- 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



Core and VPU



Flops/s, AVX, AVX2 and AVX-512
2013
2014
2015
2016
H1
H2
H1
H2
H1
H2
H1
H2
Ivy Bridge-EP
Haswell-EP
future
future
256b AVX1
16 SP / 8 DP
Flops/Cycle
256b AVX2
32 SP / 16 DP
Flops/Cycle (FMA)
512b AVX-512
Server: 64SP / 32 DP
Client: 32 SP / 16 DP
Flops/Cycle (FMA)
AVX
256-bit basic FP
16 registers
NDS (and AVX128)
Improved blend
MASKMOV
Implicit unaligned
AVX2
Float16 (IVB 2012)
256-bit FP FMA
256-bit integer
PERMD
Gather
AVX-512
512-bit FP/Integer
32 registers
8 mask registers
Embedded rounding
Embedded broadcast
Scalar/SSE/AVX "promotions"
Native media additions
Transcendental support
Gather/Scatter
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Optimization Notice
intel
23

AVX-512 Subsets [1]

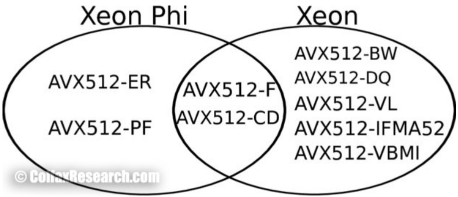
AVX-512F	Foundation instructions common between MIC and Xeon Comprehensive vector extension for HPC and enterprise All the key AVX-512 features: masking, broadcast... 32-bit and 64-bit integer and floating-point instructions Promotion of many AVX and AVX2 instructions to AVX-512 Many new instructions added to accelerate HPC workloads
AAVX-512CD	Conflict Detection instructions Allow vectorization of loops with possible address conflict Will show up on Xeon
AVX-512ER AVX-512PR	extensions for exponential and prefetch operations fast (28 bit) instructions for exponential and reciprocal and transcendentals (as well as RSQRT) New prefetch instructions: gather/scatter prefetches and PREFETCHWT1

AVX-512 Subsets [2]

AVX-512DQ	Double and Quad word instrunctions
	All of (packed) 32bit/64 bit operations AVX-512F doesn't provide Close 64bit gaps like VPMULLQ : packed 64x64 $\rightarrow$ 64 Extend mask architecture to word and byte (to handle vectors) Packed/Scalar converts of signed/unsigned to SP/DP
AVX-512BW	Byte and Word instructions
	Extent packed (vector) instructions to byte and word (16 and 8 bit) datatype MMX/SSE2/AVX2 re-promoted to AVX512 semantics Mask operations extended to 32/64 bits to adapt to number of objects in 512bit Permute architecture extended to words (VPERMW, VPERMI2W, ...)
AVX-512VL	Vector Length extensions
	Vector length orthogonality Support for 128 and 256 bits instead of full 512 bit Not a new instruction set but an attribute of existing 512bit instructions

KNL and future Xeon

- KNL and future Xeon architecture share a large set of instructions
- but sets are not identical

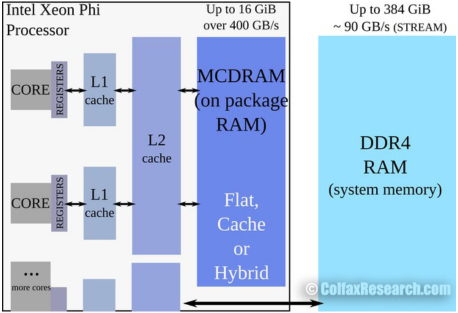


- AVX512-IFMA provides fused multiply-add instructions for 52-bit integers
- AVX512-VBMI provides additional instructions for byte-permutation and bit-manipulation.

option	to generate	from version
-xcommon-avx512	AVX-512F and AVX-512CD	15.0.2
-xmic-avx12	AVX-512F, AVX-512CD, AVX-512ER and AVX-512FP	14.0
-xcore-avx512	AVX-512F, AVX-512CD, AVX-512BW, AVX-512DQ and AVX-512VL	15.0.1

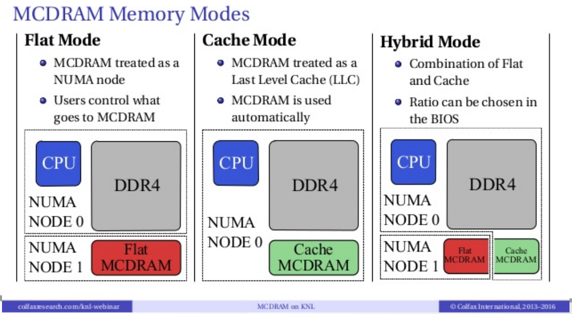
KNL Memory:MCDRAM

- Memory bandwidth in HPC is one of common bottleneck for perfomances
- To increase the demand for memory bandwidth KNL have a on-package high memory bandwidht memory (HBM) based on multi-channel dynamic random access memory (MCDRAM).
- This memory is capable of delivering up to 5x perfomance ($\geq$ 400 Gb/s) compared to DDR4 memory on same platform ($\geq$ 90 GB/s)

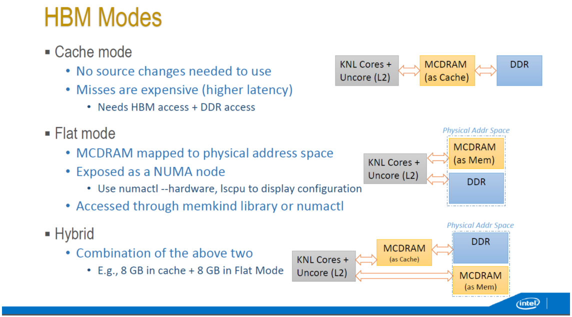


KNL Memory:MCDRAM

- HBM on KNL can be used as
  - a last-level cache
  - as addressable memory.
- The configuration is determined at boot time, by choosing in BIOS setting between three MCDRAM modes:
  - Flat mode
  - Cache mode
  - Hybrid mode



KNL Memory:MCDRAM



- The best mode to use will depend on the application.

Using HBM as addressable memory

Two methods for this:

- the numactl tool
  - Works best if the whole app can fit in MCDRAM
- the memkind library
  - Using library calls or Compiler Directives
  - Needs source modification

Using numactl to access MCDRAM

- Run "numactl –hardware" to see the NUMA configuration of your system
- Look for the node with no cores.
  - If the total memory footprint of your app is smaller than the size of MCDRAM
    - ps -C myapp u
    - see RSS value

    - Use numactl to allocate all of its memory from MCDRAM
    - numactl –membind=mcdram_id myapp
    - Where mcdram_id is the ID of MCDRAM "node"
  - If the total memory footprint of your app is larger than the size of MCDRAM
  - You can still use numactl to allocate part of your app in MCDRAM
    - numactl –preferred=mcdram_id myapp
    - Allocations that don't fit into MCDRAM spills over to DDR

    - numactl –interleave=nodes myapp
    - Allocations are interleaved across all nodes

Using Memkind to access MCDRAM

- Memkind library is a user-extensible heap manager built on top of jemalloc, a C library for general-purpose memory allocation functions.
- The library is generalizable to any NUMA architecture, but for Knights Landing processors it is used primarily for manual allocation to HBM using special allocators for C/C++
- has limited support for Fortran

Using Memkind: C case

- Allocate 1000 floats from DDR

```
float *fv;
fv = (float *)malloc(sizeof(float) * 1000);
```

- Allocate 1000 floats from MCDRAM

```
float *fv;
fv = (float *)hbw_malloc(sizeof(float) * 1000);
```

Using Memkind: Fortran case

```
C Declare arrays to be dynamic
REAL, ALLOCATABLE :: A(:), B(:), C(:)
!DEC$ ATTRIBUTES FASTMEM :: A
NSIZE=1024
c
c allocate array 'A' from MCDRAM
c
ALLOCATE (A(1:NSIZE))
c
c Allocate arrays that will come from DDR
c
ALLOCATE (B(NSIZE), C(NSIZE))
```

Using MCDRAM Summary

- Do nothing
  - If DDR BW is sufficient for your app
- Use numactl to place app in MCDRAM
  - Works well if the entire app fits within MCDRAM
  - Can use numactl –preferred if app does not fit completely in MCDRAM
- Use MCDRAM cache mode
- Trivial to try; no source changes
- Use memkind API

Trends that are here to stay

- Data Parallelism
  - Lots of threads, spent on MPI ranks or OpenMP/TBB/pthreads
  - Improving support for both peak tput and modest/single thread
- Bigger, better, faster memory
  - High capacity, high bandwidth, low latency DRAM
  - Effective caching and paging Increasing support for irregular memory refs, modest tuning
- ISA innovation
  - Increasing support for vectorizatin, new usages

Evolution or Revolution ?

Incremental changes, significant gains

- Parallelization - consistent strategy
  - MPI vs OpenMP - already needed to tune and tweak
  - Less thread-level parallelism required
  - Vectorization; more opportunity , more profitable
- Enable new features with memory using
  - Access MCDRAM with special allocation
  - Blocking for MCDRAM vs just cache

KNI specific enabling

- Recompilation with -xMIC-AVX512
- Threading: more MPI ranks, 1 thread/core
- Vectorization: incresed Efficiency
- MCDRAM and memory tuning: tile, 1 GB pages

What is needed?

- Building
  - Change compiler switches in make files
- Coding
  - Parallelization: vectorization, offload
  - Memory Management: MCDRAM enumeration and memory allocation
- Tuning
  - Potentially fewer Threads: more core but less need for SMT
  - More memory more MPI ranks

Take aways

- Keep doing what you were doing for KNC and Xeon
- Some goodness comes free with a recompile
- With some extra enabling, use new MCDRAM feature