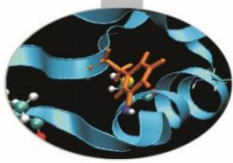
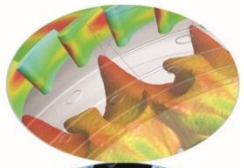
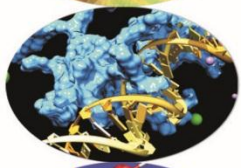
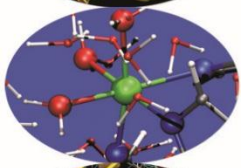
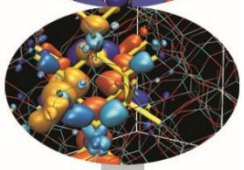
CINECA SCAI
SuperComputing Applications and Innovation



Benchmark results on Knight Landing architecture

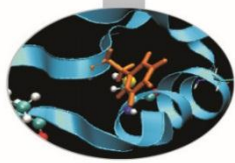





Domenico Guida, CINECA SCAI (Bologna)
Giorgio Amati, CINECA SCAI (Roma)

Milano, 21/04/2017

CINECA

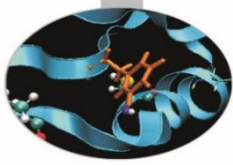
KNL vs BDW

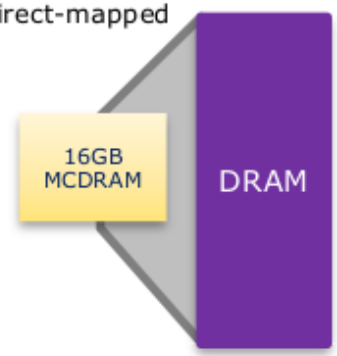
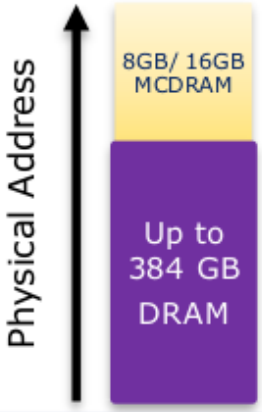
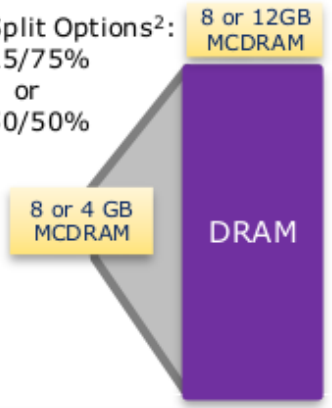


```
*********************************************************************************
* Welcome to MARCONI /                                                          *
*            MARCONI-fusion @ CINECA - NeXtScale cluster - CentOS 7.2!          *
*                                                                               *
* Broadwell partition - 1512 Compute nodes:                                     *
*    - 2 18-core Intel(R) Xeon(R) E5-2697 v4 @ 2.30GHz per Compute node         *
*    - 128 GB RAM per Compute node                                              *
* KNL partition - 3600 Compute nodes:                                           *
*    - 1 68-core Intel(R) Knights Landing @ 1.40GHz per Compute node            *
*    - 16 GB MCDRAM per Compute node                                            *
*    - 93 GB RAM per Compute node                                               *
* Intel OmniPath (100Gb/s) high-performance network                             *
* PBSpro 13 batch scheduler                                                     *
```

	A1 BDW	**A2 KNL**
cores per node	2 x 18 @2.3 GHz	1 x 68 @1,4 GHz
HYPERTHREADING	No	Yes → 272 «core»
MCDRAM	--	16 GB
RAM per node	128 GB	93 GB

KNL Memory Model



Cache Model	Flat Model	Hybrid Model
Ideal for large data size (>16GB) cache blocking apps	Maximum bandwidth for data reuse aware apps	Maximum flexibility for varied workloads
64B cache lines direct-mapped 		Split Options[2]: 25/75% or 50/50% 
Hardware automatically manages the MCDRAM as a "L3 cache" between CPU and external DDR memory	Manually manage how the app uses the integrated on-package memory and external DDR for peak perf	Harness the benefits of both Cache and Flat models by segmenting the integrated on-package memory
▪ App and/or data set is very large and will not fit into MCDRAM ▪ Unknown or unstructured memory access behavior	▪ App or portion of an app or data set that can be, or is needed to be "locked" into MCDRAM so it doesn't get flushed out	▪ Need to "lock" in a relatively small portion of an app or data set via the Flat model ▪ Remaining MCDRAM can then be configured as Cache

Code transparent

User should be aware of how his code is using memory

Very advanced programming

KNL Architecture Overview

x4 DMI2 to PCH
36 Lanes PCIe* Gen3 (x16, x16, x4)

ISA
Intel® Xeon® Processor Binary-Compatible (w/Broadwell)

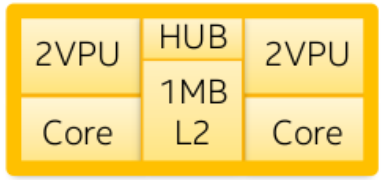
On-package memory
Up to 16GB, ~465 GB/s STREAM at launch

Platform Memory
Up to 384GB (6ch DDR4-2400 MT/s)

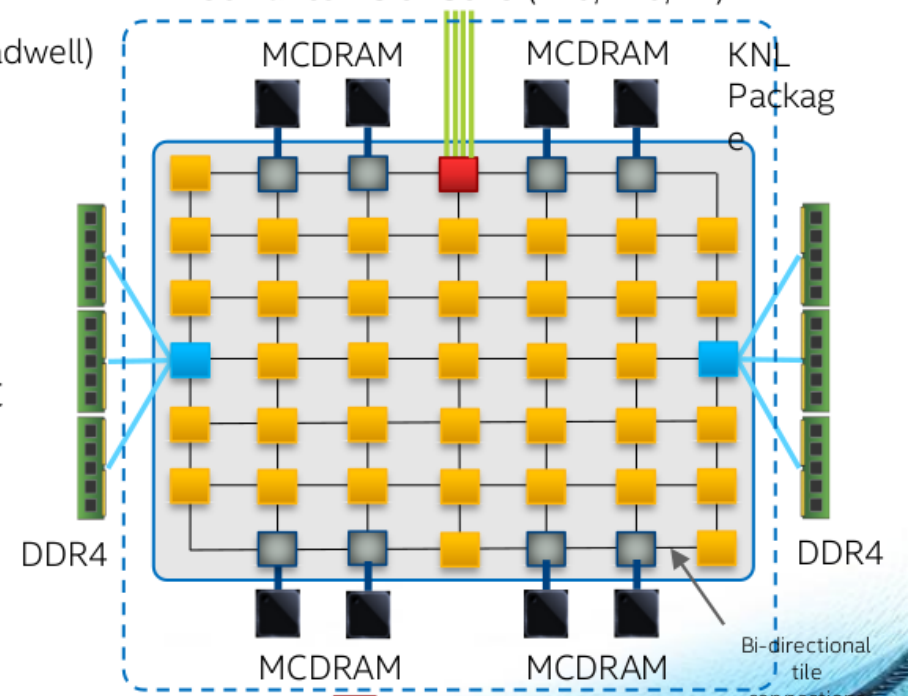
Fixed Bottlenecks

- ✓ 2D Mesh Architecture
- ✓ Out-of-Order Cores
- ✓ 3X single-thread vs. KNC

TILE: (up to 36)



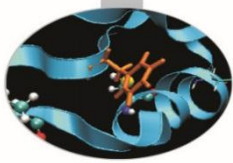
Enhanced Intel® Atom™ cores based on Silvermont Microarchitecture



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

SOFTWARE AND SERVICES

With this number of physical cores, it is important to control how OS assigns MPI jobs and OMP threads to physical cores.



AFFINITY

Processor affinity, or CPU pinning enables the binding and unbinding of a process or a thread to a central processing unit (CPU) or a range of CPUs, so that the process or thread will execute only on the designated CPU or CPUs rather than any CPU.

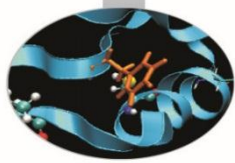
`export KMP_AFFINITY=`…

- **none (default):** the OS decides how to assign threads looking at the machine topology. (of course, OS can't really know how our code works)
- **`compact:`** <n+1>-th thread is assigned to a context as close as possible to the thread context.
- **`scatter:`** threads are distributed as evenly as possible across the entire system.

More configurations (i.e. balanced, disabled, explicit, logical, physical) are available…

https://software.intel.com/en-us/node/522691#KMP_AFFINITY_ENVIRONMENT_VARIABLE

Linpack on KNL

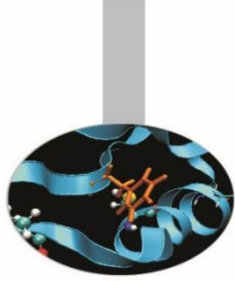


Floating point unit benchmark, OpenMP version, Intel build (mkl 2017)

- ✓ Size = 40.000, 5 replicas
- ✓ Double precision
- ✓ 64 threads
- ✓ **`KMP_AFFINITY=scatter`**

	Size	TFlops
KNL	40’000	1.9
BDW	45’000	1.3

Stream on KNL



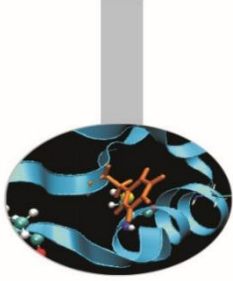
Memory Bandwidth benchmark:

- 64 threads, **`KMP_AFFINITY=scatter`**
- ✓ MCDRAM+DRAM: **`numactl –preferred 1`**
- ✓ MCDRAM/DRAM: **`numactl –membind 0/1`**
- ✓ Cache: no numactl

MEMORY	SIZE	Mb/sec
MCDRAM (flat)	9 GB	413712
MCDRAM+DRAM (flat)	23 GB	159672
MCDRAM+DRAM (flat)	90 GB	97866
DRAM (flat)	9 GB	82628
DRAM (flat)	23 GB	82507
DRAM (flat)	90 GB	82685
CACHE (cache)	9 GB	205807
CACHE (cache)	23 GB	92714
CACHE (cache)	90 GB	57882

BDW: 100’000

Molecular Dynamics codes



Gromacs

Kir3.1 potassium channel: 365K atoms, 300K, PME, Cut-off=1.2nm

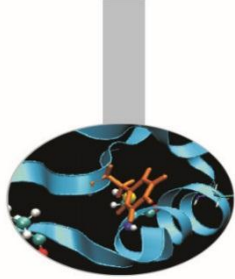
BDW	KNL (64 * 2 threads)	KNL (64 * 4 threads)
6,197 ns/day	5,8 ns/day	5,4 ns/day

Amber

Cellulose, NVT, Cut-off=0.8 nm, 400K atoms

BDW	KNL (68 *1 thread)
4,15 ns/day	5,8 ns/day

Molecular Dynamics codes



NAMD

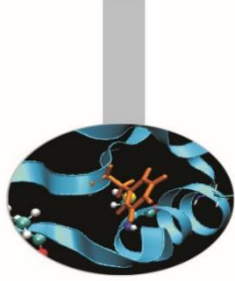
Apoa1, NPT ensemble, PME, Cut-off = 1.2 nm, 92K atoms

BDW	KNL (68 *1 thread)
1,33 ns/day	1,54 ns/day

Considerations:

Not all the codes can exploit hyper-threading on KNL (at least in their original versions).

In some cases the use of hyper-threading can improve performances of a code, in some cases it can not.



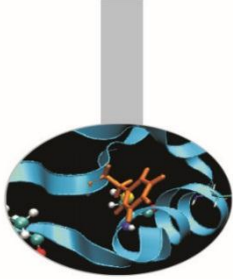
OpenFoam

We performed tests on 3 test cases with different sizes.

OpenFoam compiled for BRW runs correctly on KNL, **but** is more than 5 times slower.

We will present only results for KMP_AFFINITY=compact: it gave ~ 10% faster results than others.

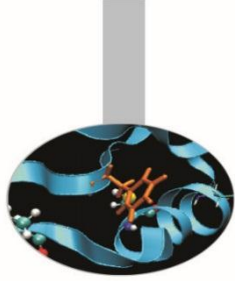
OpenFoam



3D Lid Driven Cavity Flow, size: 200^3

	Nodes	Task per node	Total time (s)
KNL	1	64	853
	1	128	777
	2	64	489
	2	128	462
	4	**64**	**267**
	4	128	386
	8	64	161
BDW	1	32	1433
	2	32	654
	4	**32**	**279**

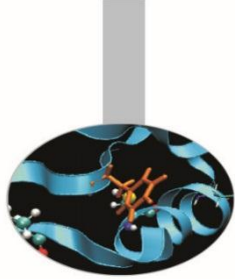
OpenFoam



3D Lid Driven Cavity Flow, size: 300^3

	Nodes	Task per node	Total time (s)
KNL	4	64	1202
	4	128	1370
	8	64	669
	8	128	1958
	16	**64**	**387**
BDW	4	32	1875
	8	32	837
	16	**32**	**384**

SpecFem3D

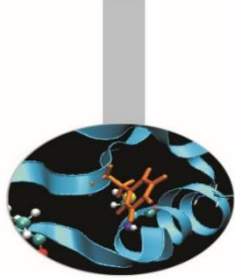


Cartesian simulation of acoustic, elastic, coupled acoustic/elastic, poroelastic or seismic wave propagation in any type of conforming mesh of hexahedra).

SpecFem3D is an hybrid code (MPI+OpenMP).

- ✓ test case: SPECFEM_GLOBE regional_MiddleEast
- ✓ quadrant/cache nodes

SpecFem3D



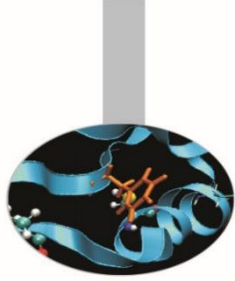
# of threads

# of cores	8	16	32	64	128	256
4	833	412	220	121	74	76
8	811	416	214	113	63	64
16		425	219	112	60	59
32			219	113	58	56
64				113	59	49
128					65	

It looks as SpecFem3D reaches the best performances using a 1 MPI to 4 OpenMP ratio.

`KMP_AFFINITY=scatter` has been used

QuantumEspresso



Tests performed to verify strong scaling.

QE is a hybrid code, using MPI as well as OpenMP threads.

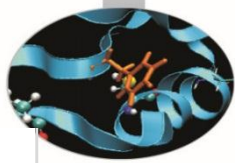
For all tests, KMP_AFFINITY=scatter has been used.

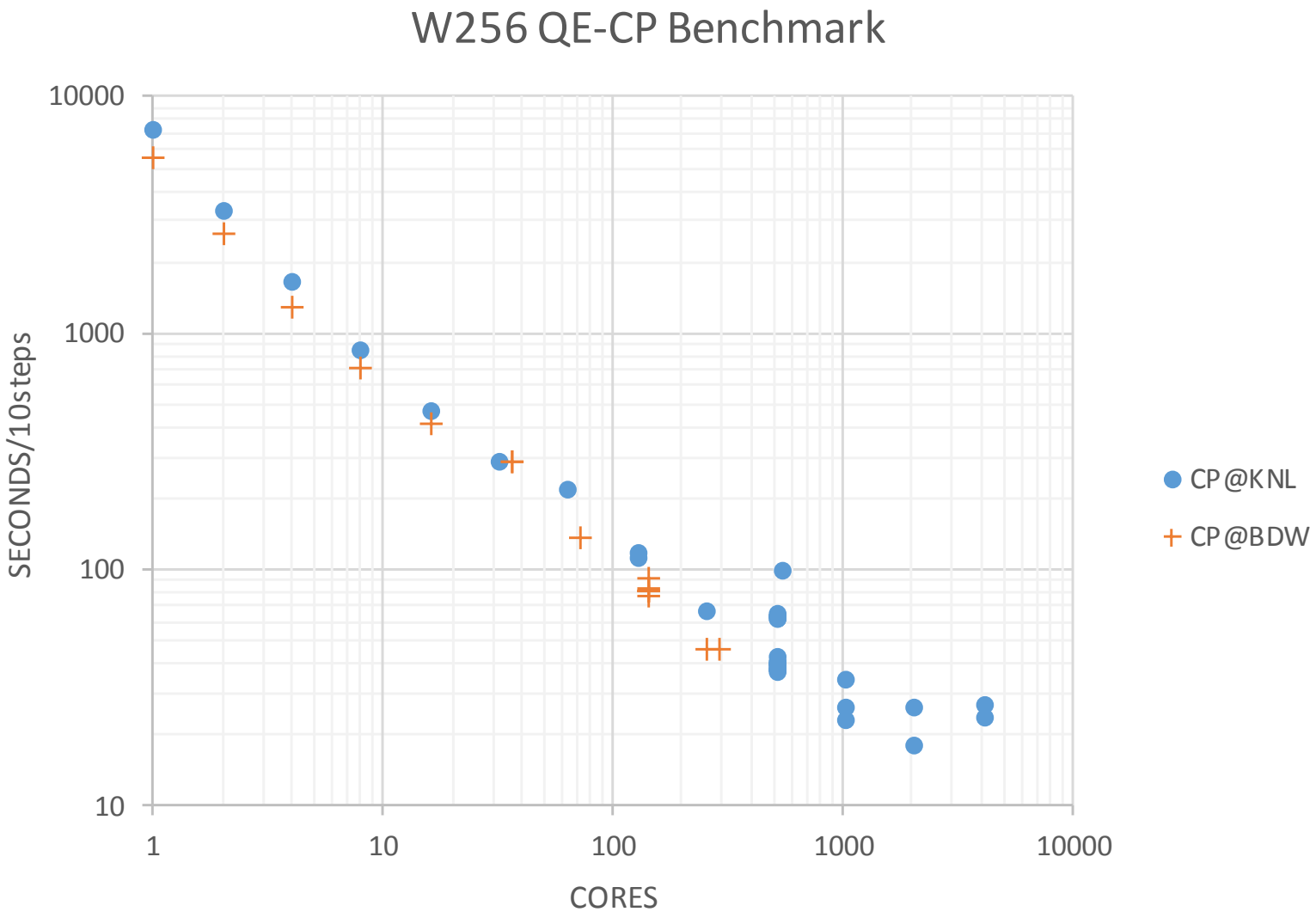
Next slides focus on computing performance vs:

- ✓ Number of cores
- ✓ Power consumption

For each of the three cases, results are compared to BDW

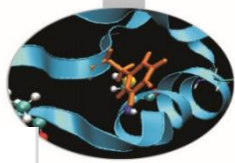
QuantumEspresso

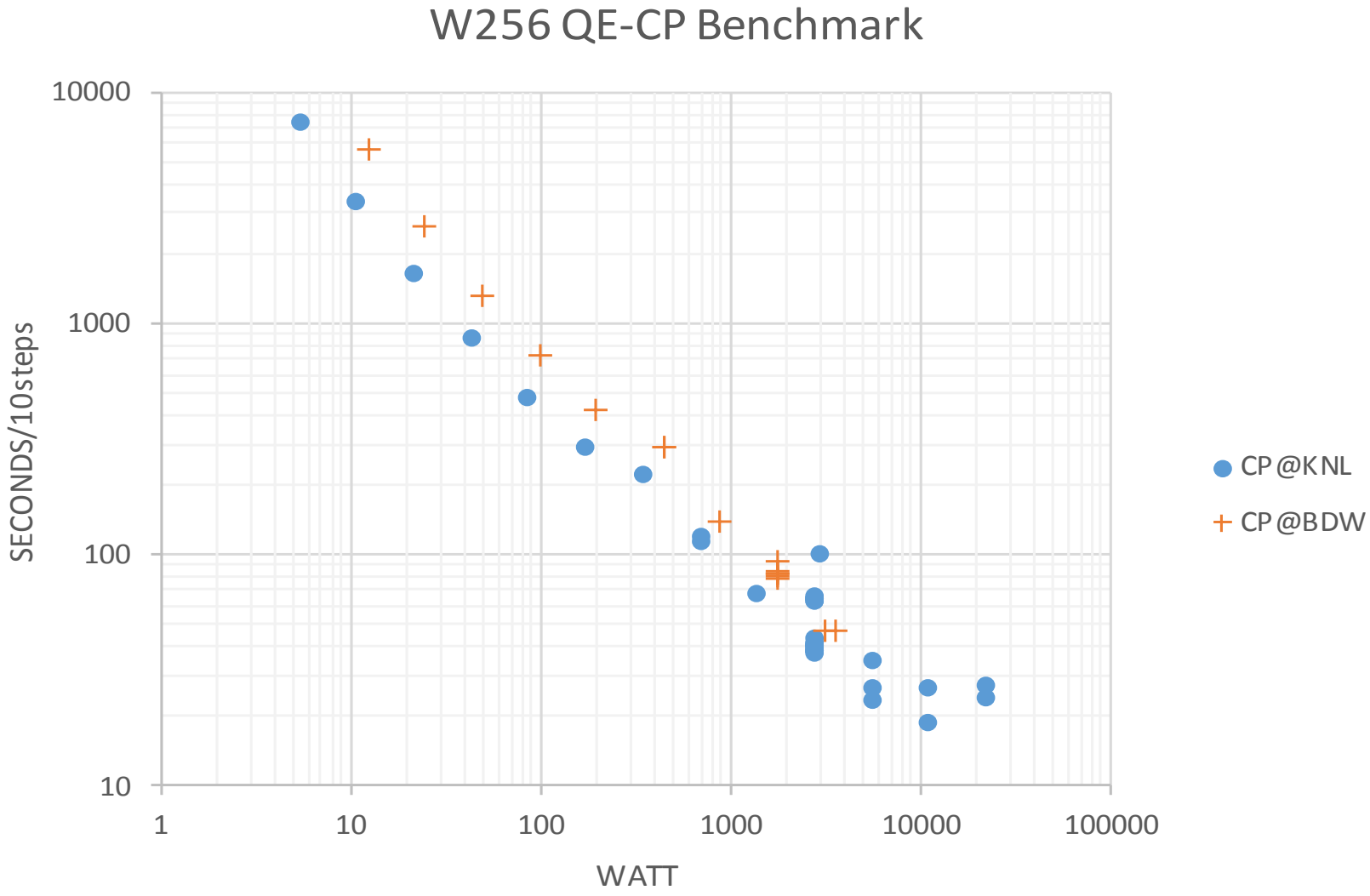




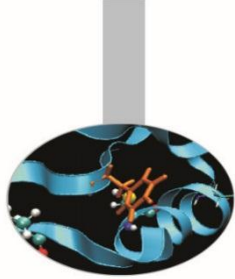
CINECA

QuantumEspresso





Hints #1



Always check vectorization

✓ KNL FPU → 512bit wide → 8 DP Flop

With no vectorization:

✓ A decrease of 8 using double precision

✓ A decrease of 16 using single precision

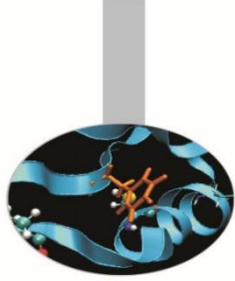
Example (3D CFD code):

- With vectorization on: 850 Mlups
- With vectorization off: 165 Mlups

Always check vectorization level with

- -qopt-report-phase=vec (*.oprpt file are generated)

Hints #2



KNL is backward compatible with BDW but

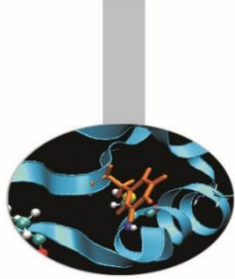
- ✓ BDW FPU → 256bit wide…
- ✓ KNL FPU → 512bit wide

If you don't recompile your code you can loose a factor 2 in performance (if the code is vectorizable)

Example (3D CFD code):

- KNL compiled code: 850 MLups
- BDW compiled code: 447 MLups

Hints #3



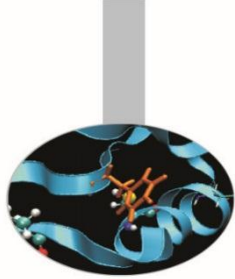
Performance (especially for OpenMP application) are really concerned to affinity.

Example (3D CFD code):

- KMP_AFFINITY=scatter: 850 Mlups
- KMP_AFFINITY=balanced: 761 Mlups
- KMP_AFFINITY=compact: 151 Mlups

Gain/Loss can be really sensitive: always play with affinity

Hints #4



Always try to exploit MCDRAM effect

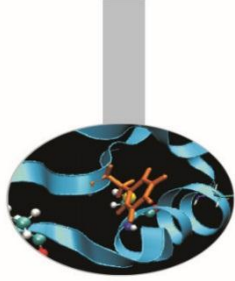
Example (3D CFD code):

- Only MCDRAM (<16 GB) : 850 Mlups
- Only DRAM (<16 GB): 372 Mlups

- MCDRAM+DRAM (20 GB): 757 Mlups
- Only DRAM (20 GB): 355 Mlups
- Cache: 523

Results are application dependent…

Hints #5

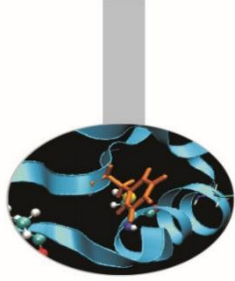


Size matters, check which is the best size-problem for your applicati!

Example (3D CFD code,2 task, 32threads, scatter):

- 128^3 → 345 Mlups
- 192^3 → 472 Mlups
- 256^3 → 535 Mlups
- 384^3 → 610 Mlups
- 512^3 → 676 Mlups
- 768^3 → 437 Mlups

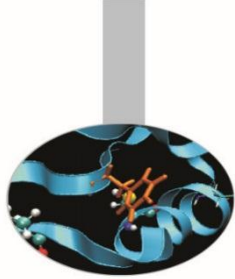
Useful links



- MARCONI User Guide: https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.1%3A+MARCONI+UserGuide
- Intel Xeon-Phi Guide (and benchmarks): https://www.aspsys.com/images/solutions/linux-cluster-solutions/knights-landing-clusters/xeon-phi-knl-marketing-guide.pdf
- Guide to vectorization: https://wiki.u-gov.it/confluence/display/SCAIUS/How+to+Improve+Code+Vectorization

For any kind of support, please refer to superc@cineca.it : your request will be assigned to someone who can understand your problem and give you support.

Aknwoldgments



- ✓ The User Support team (superc@cineca.it)
- ✓ Piero Lanucara
- ✓ Alessandro Grottesi
- ✓ Mariella Ippolito