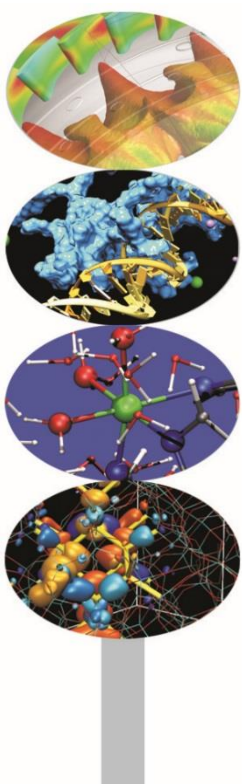
Benchmark results on Knight Landing (KNL) architecture

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

Roma 23/10/2017



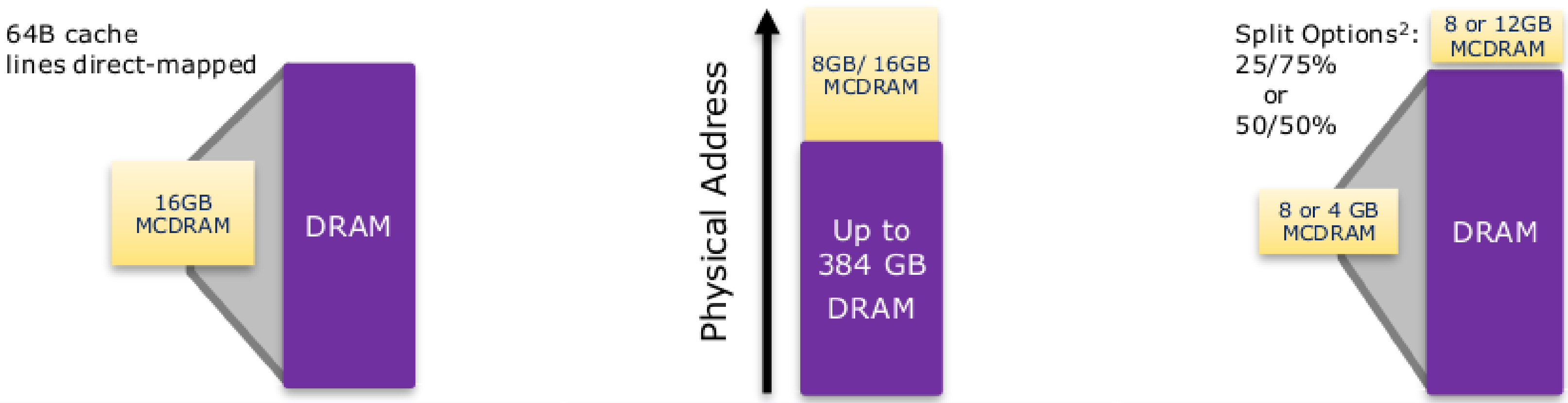
KNL, BDW, SKL

```
***************************************************************************************
* Welcome to MARCONI /
*            MARCONI-fusion @ CINECA - NeXtScale cluster - CentOS 7.2!
*
* Broadwell partition - 1512 Compute nodes with:
*     - 2*18-core Intel(R) Xeon(R) E5-2697 v4 @ 2.30GHz
*     - 128 GB RAM
* KNL partition - 3600 Compute nodes with:
*     - 1*68-core Intel(R) Knights Landing @ 1.40GHz
*     - 16 GB MCDRAM + 93 GB RAM
* SKL partition - 1512+792 nodes with:
*     - 2*24-core Intel Xeon 8160 CPU @ 2.10GHz
*     - 192 GB DDR4 RAM
* Intel OmniPath (100Gb/s) high-performance network
* PBSpro 13 batch scheduler
*
```

	A1 BDW	A2 KNL	A3 SKL
cores per node	2 x 18 @2.3 GHz	1 x 68 @1,4 GHz	2 x 24@2.1 GHz
HYPERTHREADING	No	Yes → 272 «core»	No
MCDRAM	--	16 GB	--
RAM per node	128 GB	93 GB	192 GB

KNL Memory Model

Cache Model Ideal for large data size (>16GB) cache blocking apps	Flat Model Maximum bandwidth for data reuse aware apps	Hybrid Model Maximum flexibility for varied workloads



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

ISA

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

On-package memory

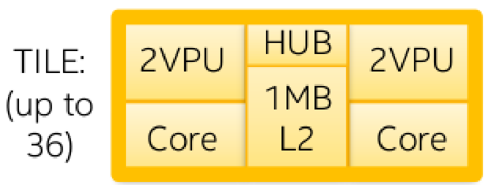
Up to 16GB, ~465 GB/s STREAM at launch

Platform Memory

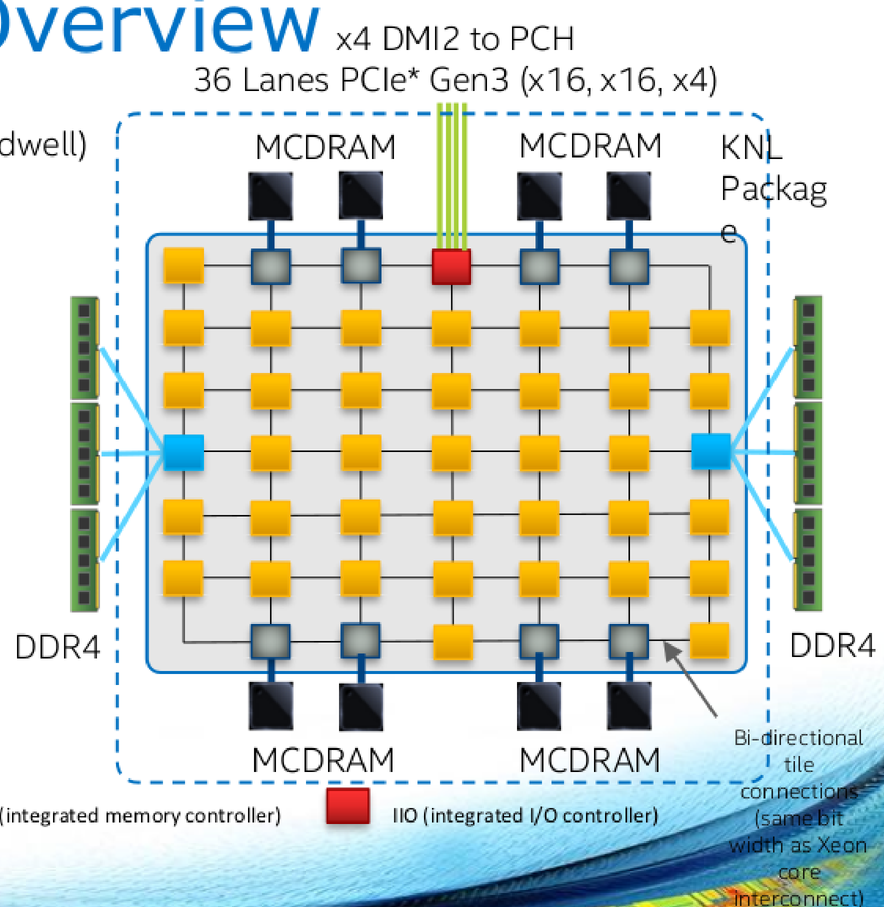
Up to 384GB (6ch DDR4-2400 MT/s)

Fixed Bottlenecks

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



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 binding-unbinding of a process-thread to a central processing unit (CPU) or a range of CPUs, so that the process-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)

- ✓ 5 replicas
- ✓ Double precision
- ✓ Reference value, not the best
- ✓ **`KMP_AFFINITY=scatter`**

	Threads	Size	TFlops
SKL	48	45’000	2.0
KNL	64	40’000	1.9
BDW	36	45’000	1.3

Stream on KNL

Memory Bandwidth benchmark:

- 64 threads, **`KMP_AFFINITY=scatter`**
- ✓ MCDRAM+DRAM: **`numactl –preferred 1`**
- ✓ MCDRAM/DRAM: **`numactl –membind 1/0`**
- ✓ 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

Reference:
BDW: 100’000

Technological trends

Some figures about Intel CPU evolution @CINECA (2010-2017)

- ✓ No more clock increase
- ✓ Total number of core increase → x4 SKL, x6 KNL
- ✓ Flops/cycle ratio increase → x8

CPU (codename)	Clock Frequency	Number of core	Flops /cycle (DP)	Peak Perf. (DP)
Xeon E5645 (Westmere)	2.4 GHz	2x6	4	115 GFlops
Xeon E5-2687W0 (Sandy Bridge)	3.1 GHz	2x8	8	396 GFlops
Xeon E5-2670v2 (Ivy Bridge)	2.5 GHz	2x10	8	400 GFlops
Xeon E5-2630v3 (Hashwell)	2.4 GHz	2x8	16 (AVX-256bit)	614 GFlops
Xeon E5-2697v4 (Broadwell)	2.3 GHz	2x18	16 (AVX-256bit)	1325 GFlops
Xean Phi (Knights Landing)	1.4 Ghz	1x68	32 (AVX-512bit)	3046 GFlops
Xeon Platinum (Skylake)	2.1 GHz	2x24	32 (AVX-512bit)	3225 GFlops

How to exploit performance

- Today single core performance is not an issue
- Multi-core CPU performance is the main issue
- Serial performance tends to be meaningless
- Single node/CPU is a meaningful figure

- To exploit CPU performance it is mandatory
  - ✓ Parallelism → factor 4 in about 10 years
  - ✓ Vectorization → factor 8 in about 10 years

This is the actual HPC evolution: if you are not ready to implement these issue you'll never reach the claimed performance!!!!

Hints #1

Always check for vectorization

✓ KNL FPU → 512bit wide → 8 DP Flop

No vectorization means:

✓ a possible decrease of a factor 8 using double precision

✓ a possible decrease of a factor 16 using single precision

Example (3D CFD code, Higher is better):

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

Always check vectorization level with

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

Hints #2

KNL is backward compatible with BDW but
- ✓ BDW FPU → 256bit wide
- ✓ KNL FPU → 512bit wide
- ✓ SKL FPU → 512bit wide

If you don’t recompile your code you can loose a factor 2 in performance (if the code can be vectorized)

Example running on KNL (3D CFD code, Higher is better):
- With KNL-compiled code: 850 MLups
- With BDW-compiled code: 447 MLups

Marconi front-end node are BDW based!!!

Hints #3

Performance (for OpenMP application) are really sensible to affinity.

Example (3D CFD code, Higher is better):

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

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

SuperComputing Applications and Innovation

Hints #4

Size matters, check which is the best size/problem for your application…

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

Hints #5.1

- CFD code
- **Writing** time (sec.) on disc: formatted vs. unformatted
- BDW vs KNL
- They share the same filesystem

	size	BDW	KNL	Ratio
formatted	211 MB	58"	468"	8.0
binary	1200 MB	1.20"	1.25"	-

- General issues:
  - ✓ Always: avoid formatted output
  - ✓ It is even worst on KNL

Hints #5.2

- Another CFD code
- **Reading** time (sec.) formatted data from disk
- BDW vs KNL
- They share the same filesystem

	size	BDW	KNL	Ratio
formatted	3.0 GB	292"	1597"	5.5

- General issues:
  - ✓ Always: avoid formatted output
  - ✓ Parallel I/O could help (see also Hint #4)

Hints #6

exploit MCDRAM effect

Example (3D CFD code):

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

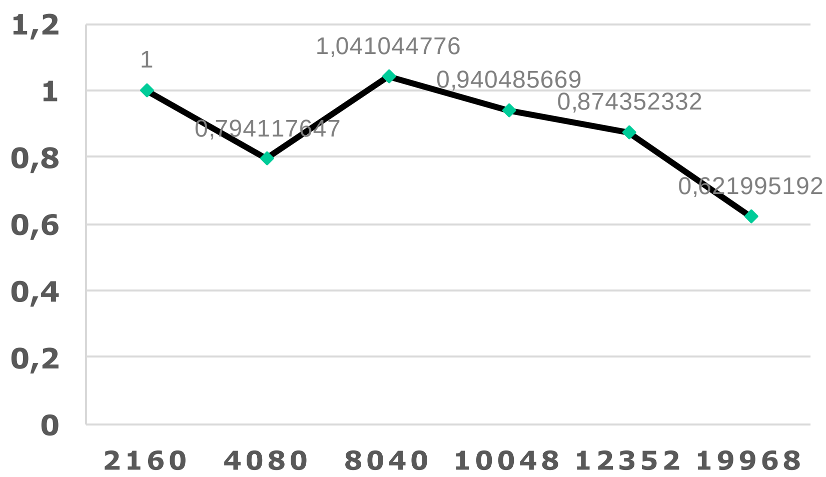
- MCDRAM+DRAM (20 GB): 757 Mlups
- Only DRAM (20 GB): 355 Mlups
- Cache: 523

Results are application/size dependent…

SuperComputing Applications and Innovation

NEMO GLOB16 benchmark

- A very high resolution version of the NEMO ocean model developed at CMCC
  - ✓ Fully MPI (scalability)
  - ✓ Highly vectorized (key_vectopt_loop) preprocessing keywords to enhance loop-level vectorization
  - ✓ XIOS2 server for efficient/scalable parallel I/O of huge files (model's output, diagnostic,…)
  - ✓ Parallel efficiency up to 19968 cores



SPECFEM3D_GLOBE benchmark/1

- Highly vectorized (FORCE_VECTORIZATION pre-processing keywords to enhance loop-level vectorization)
- Regional Greece_small benchmark (serial, only 1 thread)
  - ✓ With vectorization on: 1340 sec. (solver)
  - ✓ With vectorization off: 1883 sec. (solver)

Comment about vectorization:

- be careful with compiler options, as they may interact with vectorization: **`-fpe0`** vs **`simd`** directive?

remark #15326: simd loop was not vectorized: implied FP exception model prevents vectorization. Consider changing compiler flags and/or directives in the source to enable fast FP model and to mask FP exceptions

remark #15552: loop was not vectorized with "simd"

Always check vectorization level with

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

SPECFEM3D_GLOBE benchmark/2

- Multiple threads scaling: Fully OpenMP (1 MPI process)
- Regional_Greece_small benchmark (up to 128 OpenMP threads)
- Very good scaling up to 64 cores
- **KMP_AFFINITY** equal to scatter must be used!

# threads	Solver time	Speed-up
1	1340″	-
2	695″	1.92
4	348″	3.85
8	177″	7.57
16	91″	14.7
32	48″	27.9
64	28″	47.8
128	26″	51.5

SPECFEM3D_GLOBE benchmark/3

- Intranode scaling: Fully hybrid (MPI+OpenMP)
- Regional_Greece_small benchmark (varying number of MPI processes and OpenMP threads)

✓ Very good intranode scaling

✓ Best result achieved using both MPI processes and OpenMP threads

✓ **`KMP_AFFINITY=scatter`** must be used!

Task/Threads	Solver time
4x1	342''
4x2	180''
4x4	91''
4x8	48''
4x16	25''
4x32	22''

Task/Threads	Solver time
16x1	95''
16x2	49''
16x4	26''
16x8	24''

Task/Threads	Solver time
64x1	30''
64x2	22''

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 (36 task*1 thread)	KNL (68 task *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

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

KNL

Nodes	Task per node	Total time (s)
1	64	853
1	128	777
2	64	489
2	128	462
4	**64**	**267**
4	128	386
8	64	161

BDW

Nodes	Task per node	Total time (s)
1	32	1433
2	32	654
4	**32**	**279**

OpenFoam

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

KNL

Nodes	Task per node	Total time (s)
4	64	1202
4	128	1370
8	64	669
8	128	1958
16	**64**	**387**

BDW

Nodes	Task per node	Total time (s)
4	32	1875
8	32	837
16	**32**	**384**

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.

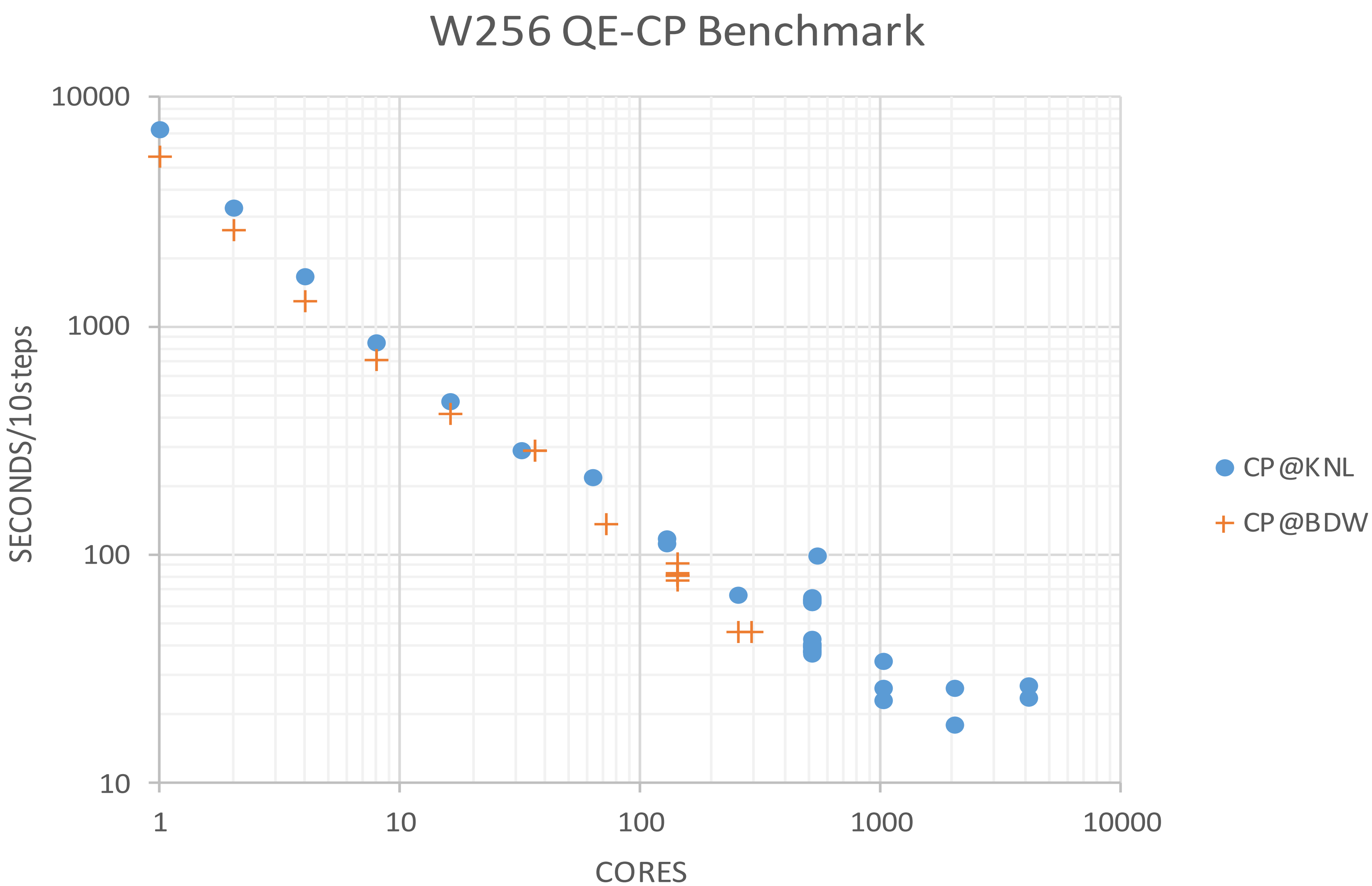
Next slides focus on computing performance vs:

- ✓ Number of cores
- ✓ Power consumption

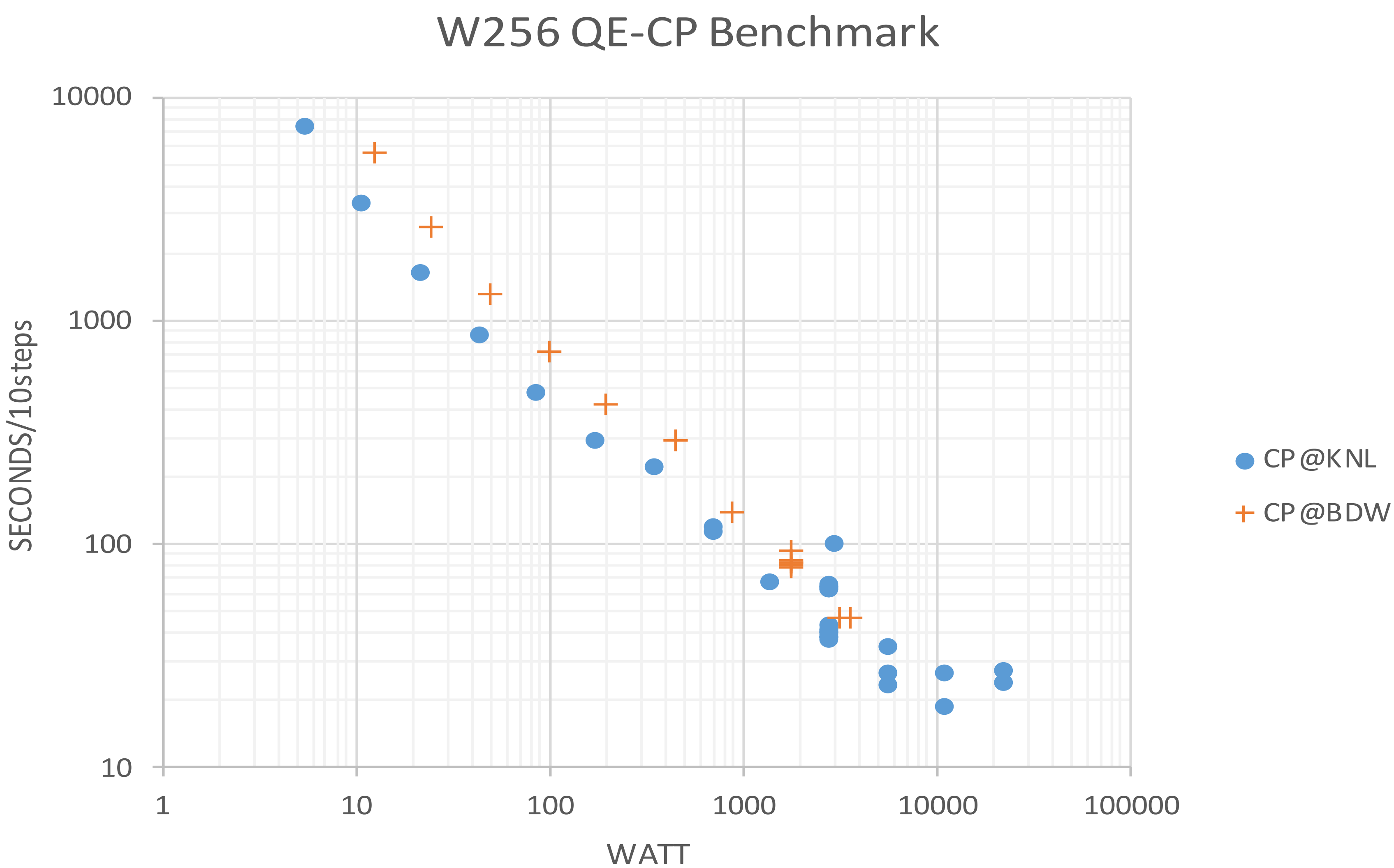
For each of the three cases, results are compared to BDW

SuperComputing Applications and Innovation

QuantumEspresso



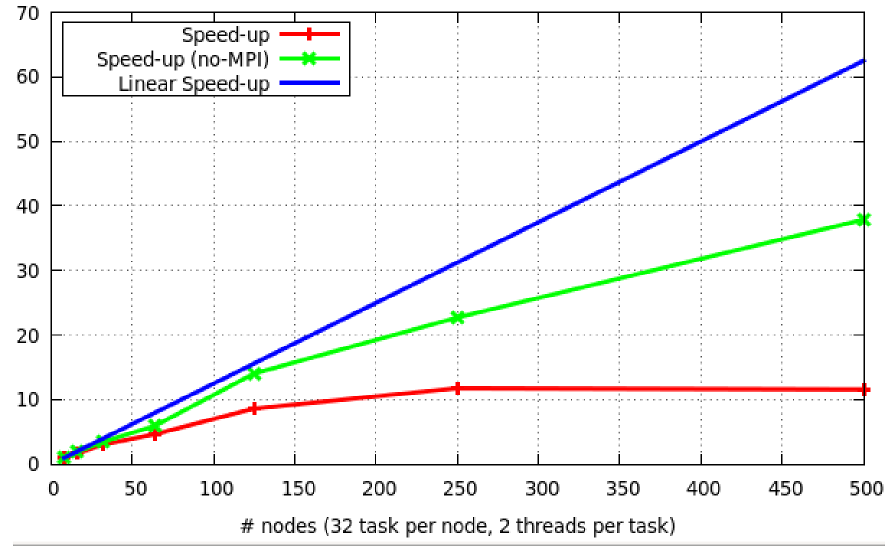
QuantumEspresso

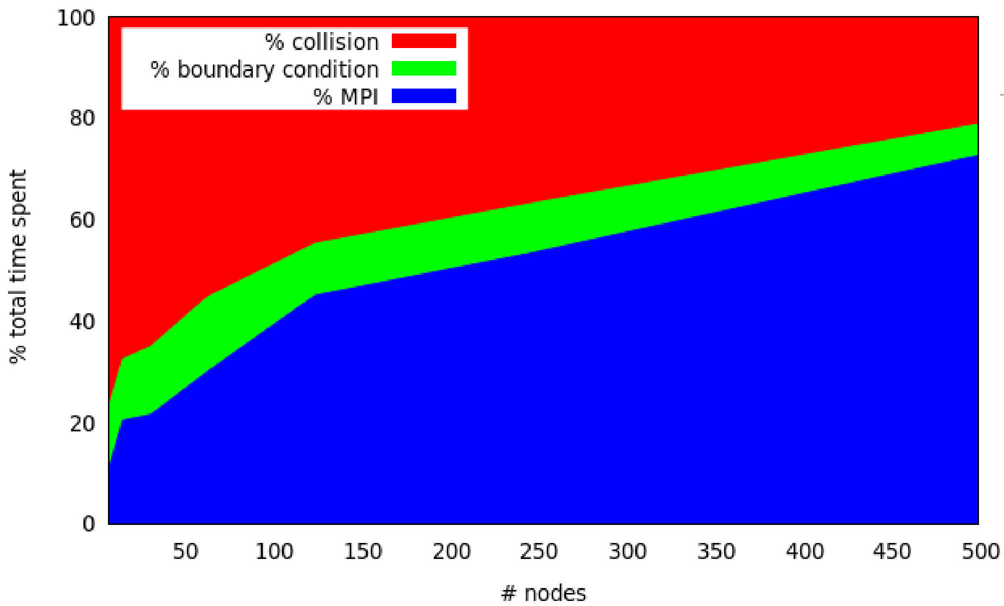


Scalability: speed-up

CFD Kinetic code (Lattice Boltzmann)

Scalability for 1200*600*200 grid

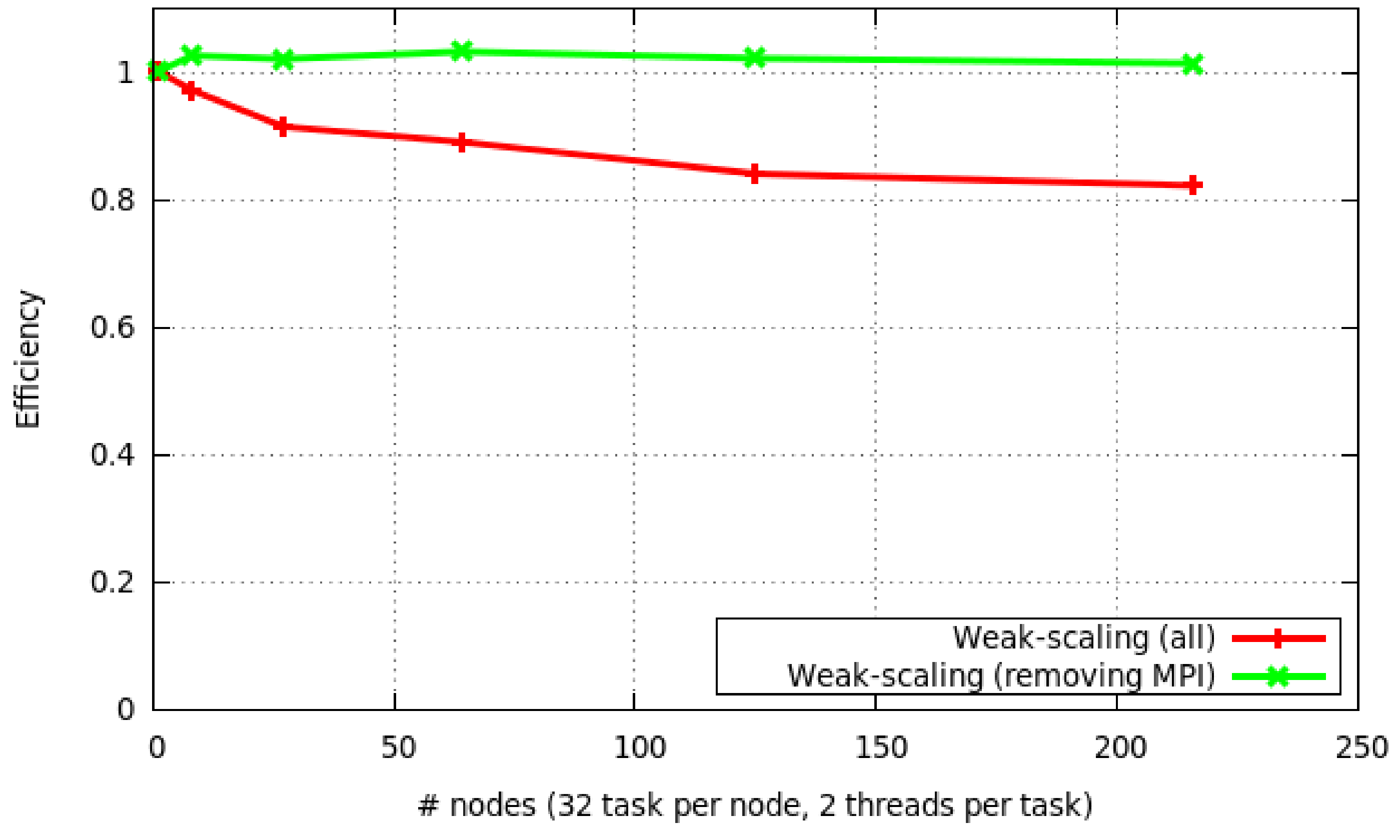




Scalability: scale-up

CFD Kinetic code (Lattice Boltzmann) scalability

✓ From 36’000’000 (1 node) to 7’760’000’000 (216 nodes) grid-points



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
  - ✓ https://hpc-forge.cineca.it/files/ScuolaCalcoloParallelo_WebDAV/public/anno-2017/26th_Summer_School_on_Parallel_Computing/Roma/VECTORIZATION-slides.pdf

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.

SuperComputing Applications and Innovation

Acknowledgments

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