CINECA

Performance Assessment and Energy Efficiency of MaX Codes

First Workshop on Software Co-Design Actions in European Flagship HPC Codes in Conjunction with ISC 2022
Hamburg, Germany
June 2, 2022

Daniele Cesarini
HPC Specialist - CINECA

Federico Ficarelli
Federico Tesser
Andrea Piserchia
Fabio Affinito

CINECA

Overview

- *Introduction to the codesign work of MaX*
- *Jump into HW performance counters*
- *A view on Linux Perf*
- *How to monitor energy/power*
- *Roofline and TMAM performance models*
- *Experimental results on MAX codes*
- *Conclusions*

Acknowledgments

MaX Project

Most of the experimental work presented in these slides was performed in the WP4 codesign work package of MaX project

Regale Project

The software stack and the power management tools was developed in the REGALE project

EPI Project

The microarchitecture analysis and the performance assessment are part of the work of EPI-SGA1/2 projects

Codesign work of MaX

We will present some of the work done in the WP4 (codesign WP) of MaX

This work was performed on conventional HPC system without using accelerators

This codesign work will focus on:

- Microarchitecture efficiency of the application workloads
- Energy efficiencies of the codes
- Performance portability
- Performance and power efficiency at scale with MPI and OpenMP

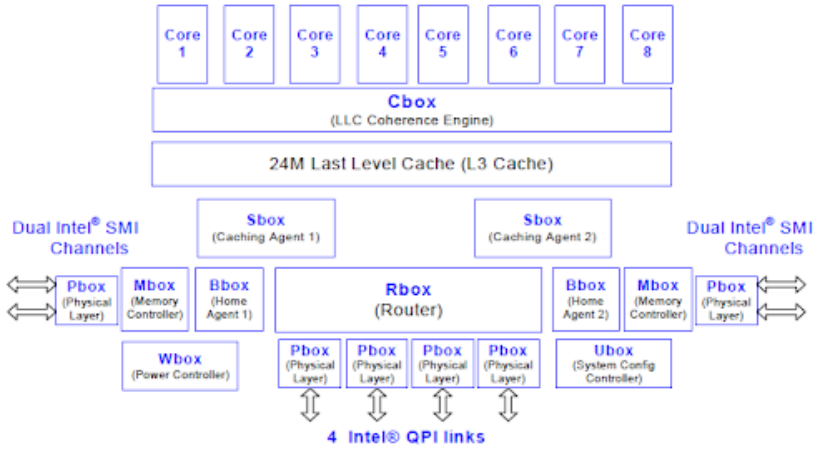
Performance Monitoring Unit (PMU)

The CPU supports you with the PMUs!

A PMU usually support many events (cycles, instructions retired, etc.) through **Performance Monitoring Counters (PMC)**.

A PMU can be:

- **on-core**: microarchitecture events at the core level (cycles, instructions retired, ...)
- **off-core:** microarchitecture events outside of cores (memory read/write, …)



PMCs can be:

- **fixed**: can be only enabled or disable and profile a specific event
- **configurable**: can monitor many events

PMCs are usually configurable only at kernels level

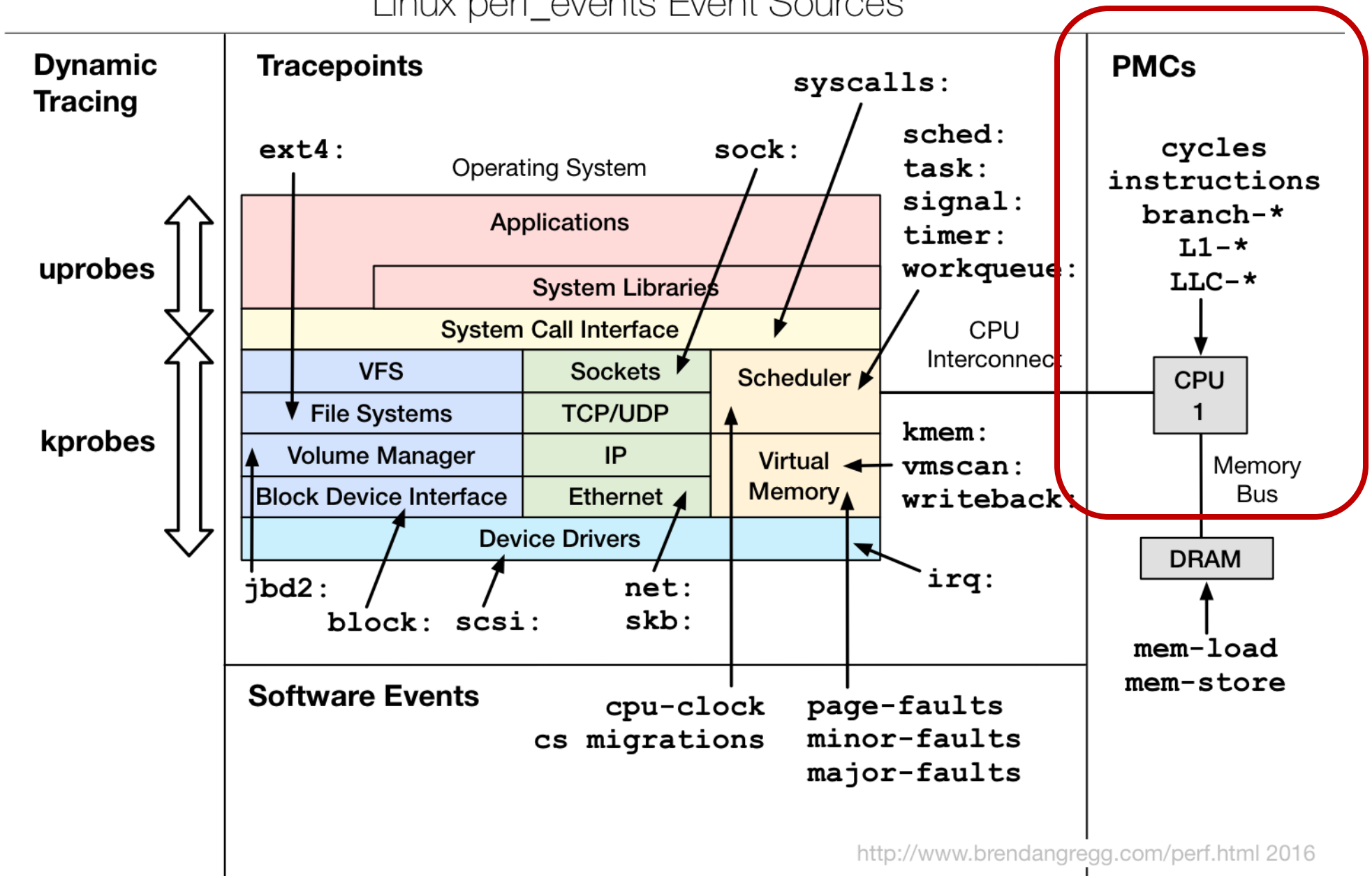
Usually, CPUs provide at user space some assembly instructions with low overhead (see *rdpmc()*) to read PMCs

Enter *Linux perf*

- *Official* Linux profiler
  - Built on top of kernel infrastructure (`ftrace`)
  - Source and docs in kernel tree
- Provides a plethora of profiling/tracing features at all system levels
  - user, kernel, CGROUP, etc...
- Most important for us: **a comprehensive toolbox to gain workload execution insights via PMCs**
- Low overhead*
  - Tunable
  - 1-2% counting mode, 5-15% sampling w/multiplexing

* Nowak, Andrzej et al. "Establishing a Base of Trust with Performance Counters for Enterprise Workloads." *USENIX Annual Technical Conference* (2015).

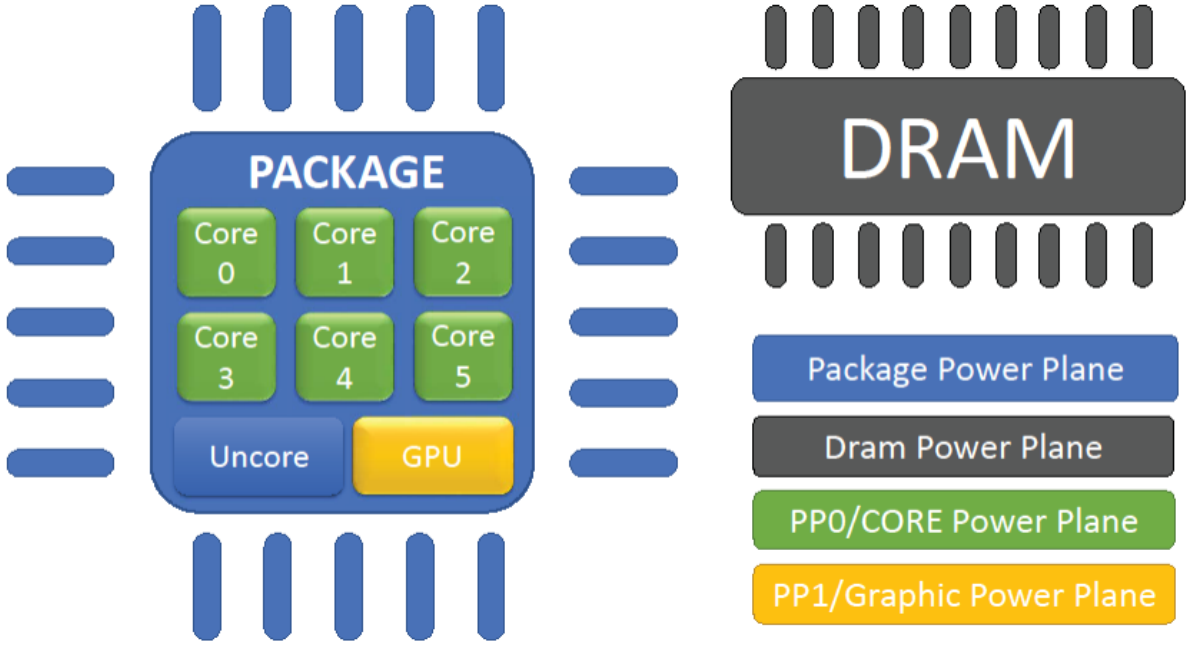
Linux perf_events Event Sources



http://www.brendangregg.com/perf.html 2016

Intel power management - RAPL

Intel CPU implements hardware power controller called Running Average Power Limit (RAPL)



Power Domains

Package Domain: monitor/limit the power consumption for the entire package of the CPU, this includes cores and uncore components.

DRAM Domain: monitor/limit power consumption of the DRAM memory. It is available only for server architectures. (no client)

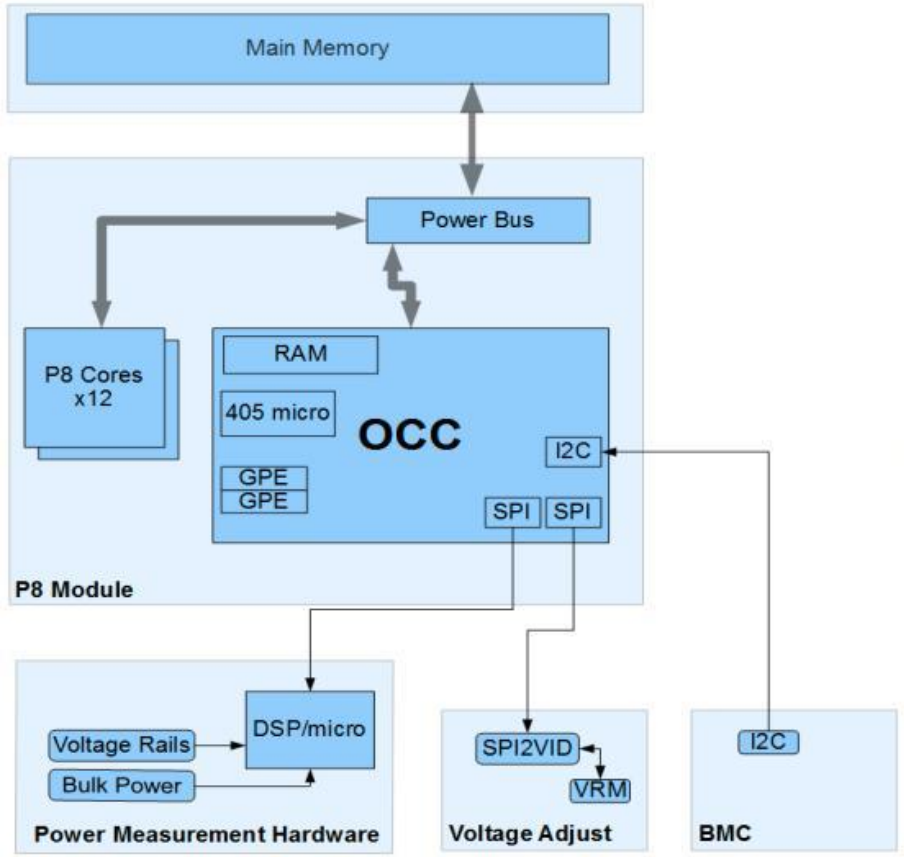
PP0/Core Domain: is used to monitor and limit the only to the cores of the CPU.

PP1/Graphic Domain: is used to monitor and limit the power only the graphic component of the CPU (no server).

Machine Specific Registers: MSR address space 0x600 – 0x640

Sysfs Interface: */sys/devices/virtual/powercap/intel-rapl/intel-rapl:X/intel-rapl:0:Y* X=socket_id, Y=power_domain

IBM Power8/9 Power Management - OCC



What is the On-Chip Controller (OCC)?

- 405 microcontroller with 512k dedicated RAM
- Hardware/Firmware that controls power, performance & thermal (independent micro OS)
- Can communicate in band with the host through the main memory

What does OCC do?

- Reads/controls system power (CPUs, GPUs, board, etc.)
- Reads/controls chip temps (CPUs, GPUs, board, etc.)
- Enables efficient fan control
- Provides thermal protection
- Power Capping
- Fault Tolerance
- Energy saving
- Performance boost

In-band: through Linux *occ-hwmon* kernel module: */sys/firmware/opal/exports/occ_inband_sensors* + C data structs that define the 405 memory address space (https://www.kernel.org/doc/html/v5.9/hwmon/occ.html, https://github.com/open-power/occ)
Out-of-band: through the BMC and IP network communication

Cavium ThunderX2 Power Management - TX2MON

Cavium ThunderX2 processor implement a similar RAPL/OCC controller

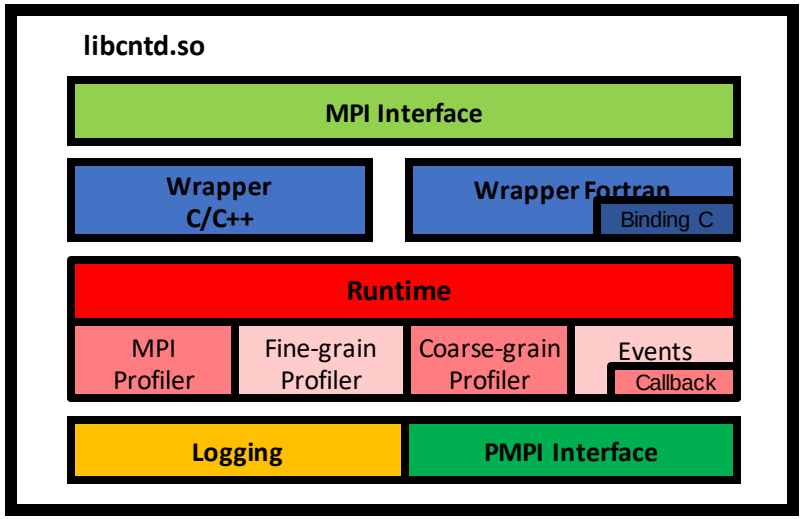
Unfortunately, we don't have several information on it, but we know that is possible to monitor the following power domains:

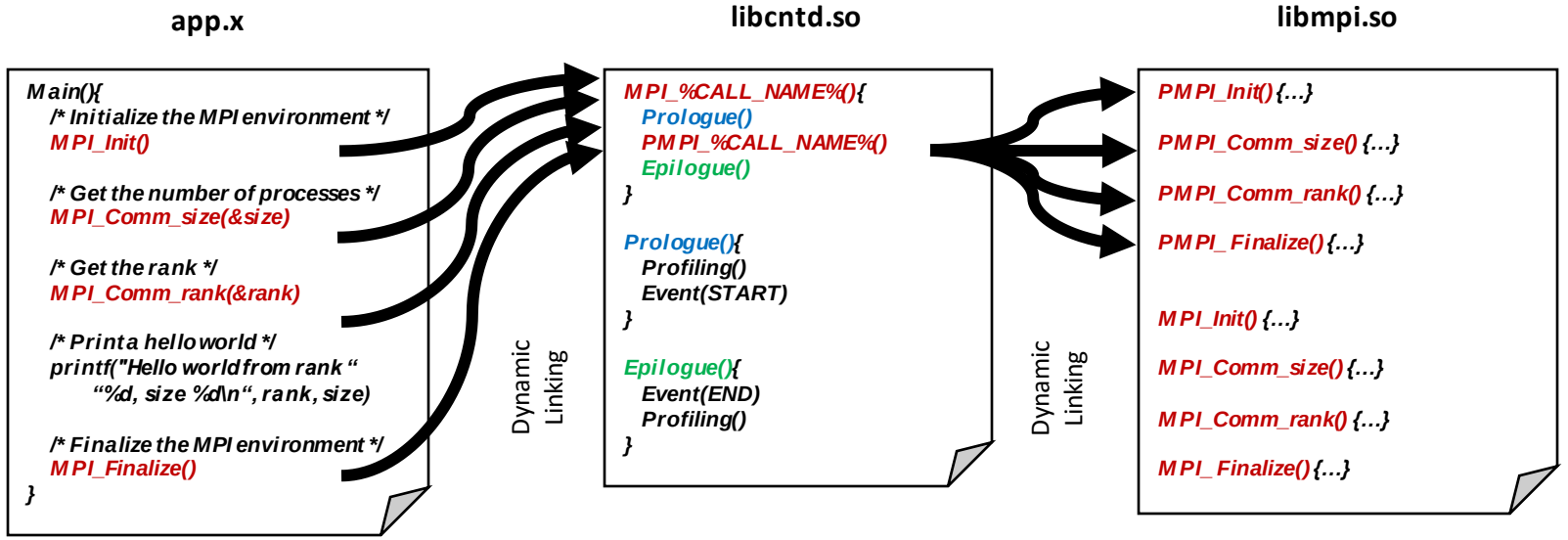
- **Core**: voltage and power consumed by all cores on the SoC.
- **SRAM**: voltage and power consumed by internal SRAM on the SoC.
- **Internal memory**: voltage and power consumed by the LLC ring on the SoC.
- **SoC**: voltage and power consumed by miscellaneous SoC blocks.

In-band: through Linux *TX2MON* kernel module: */sys/devices/platform/tx2mon* + C data structs that define the controller memory address space (https://github.com/Marvell-SPBU/tx2mon)
Out-of-band: through the BMC and IP network communication

So, how can we collect power/energy information from different systems?

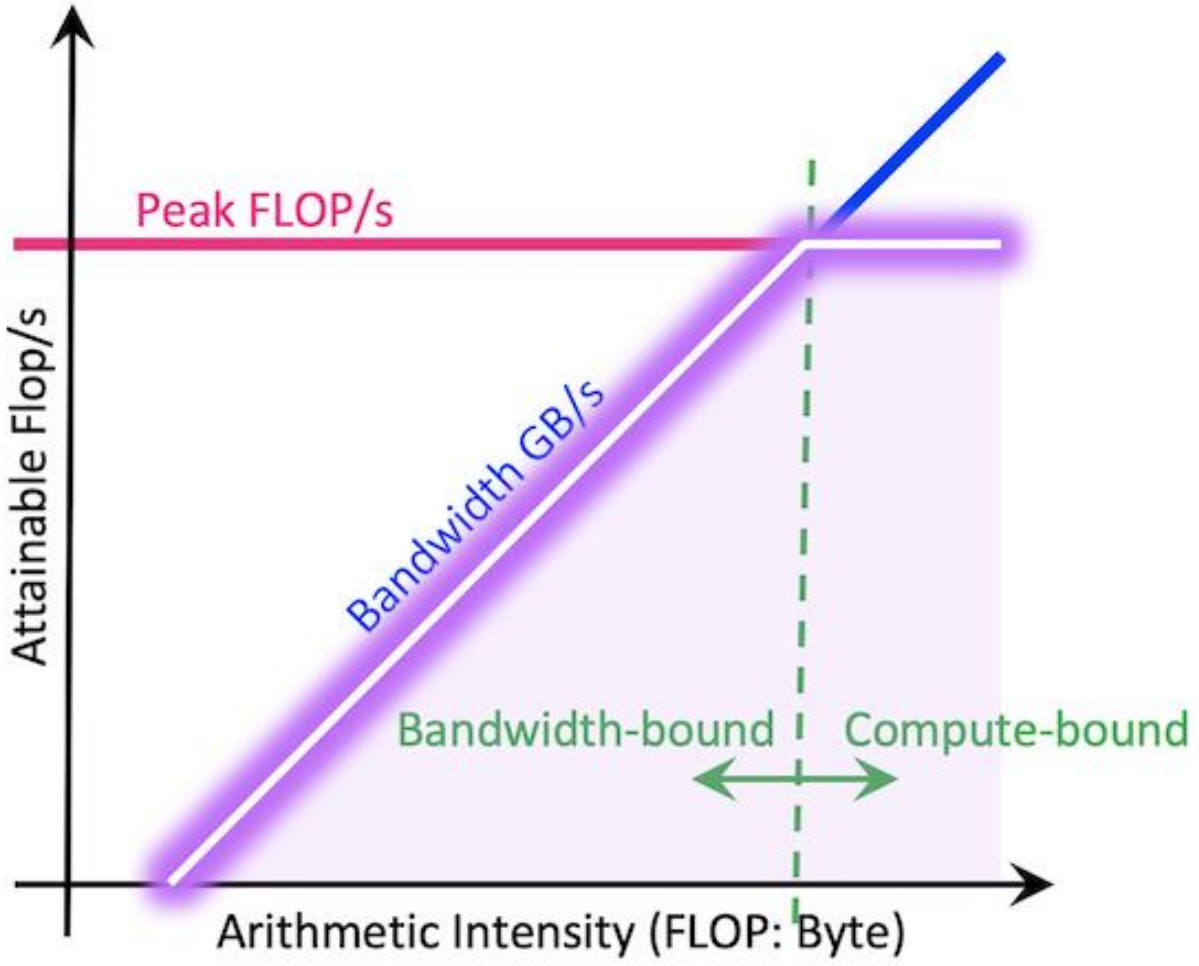
- Not easy at all!
- For this reason, we developed **COUNTDOWN** (CINECA + UNIBO project)
- **COUNTDOWN** instruments the application at execution time and collect information on the application workload and on the energy/power consumption of the system





COUTDOWN: https://github.com/EEESlab/countdown

Compute vs Data Movement - *The roofline model*

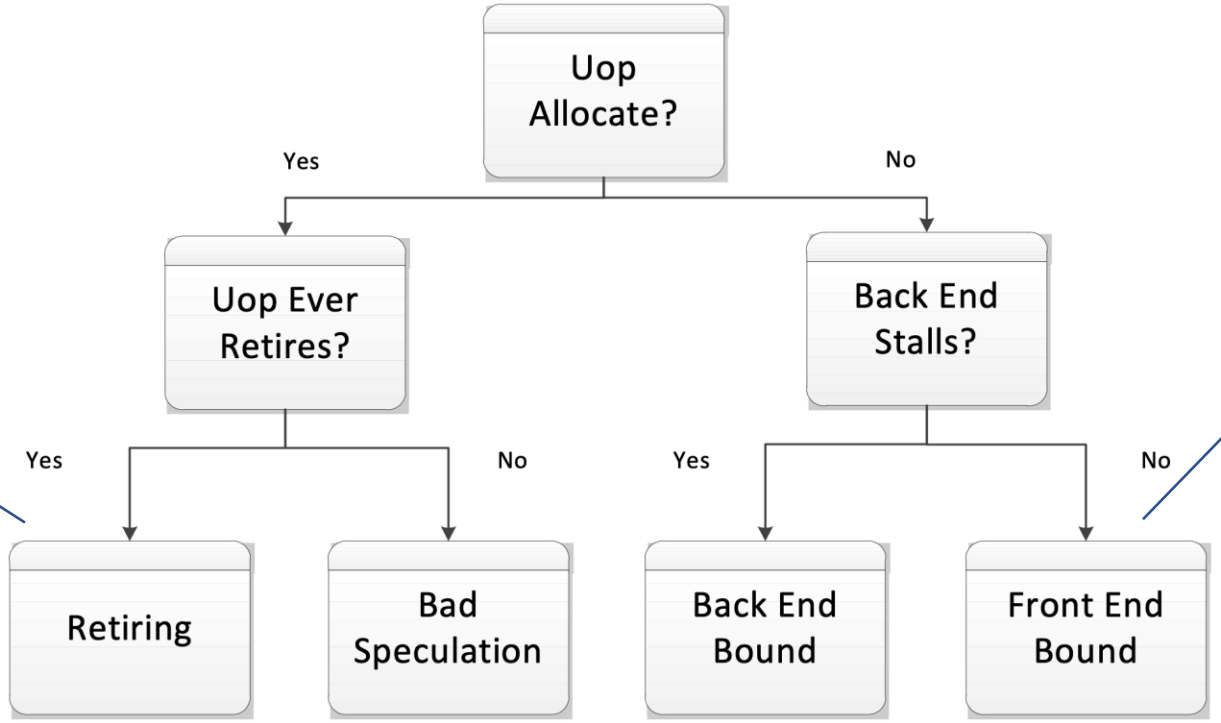


- Is performance limited by compute or data movement?
- **Y-axis:** performance in FLOPS
- **X-axis:** Arithmetic Intensity AI (FLOP/Byte)
  - Ratio between total FLOP and total byte exchange with main memory
  - Measure of data locality (data reuse)
  - Typical machine balance is 5-10 AI
  - Stream TRIAD: 0.083 AI (2 FLOP per 24 bytes)
- Application/kernel near the roofline are making **good use** of computational resources
  - Compute bound: >50% of peak performance
  - Bandwidth bound: >50% of Stream Triad
- **Bad performance**:
  - Insufficient cache bandwidth
  - Bad data locality
  - Integer heavy code
  - Lack of FMA
  - Lack vectorization

Drill down: identify bootlenecks with TMAM

- **T**op-down **M**icroarchitecture **A**nalysis **M**ethod
- first attempt by Intel at a simplified approach
- **hierarchical drill-down method guided by PMCs**
- **goal: identify the real bottleneck WRT micro-architecture**

TMAM: high level breakdown



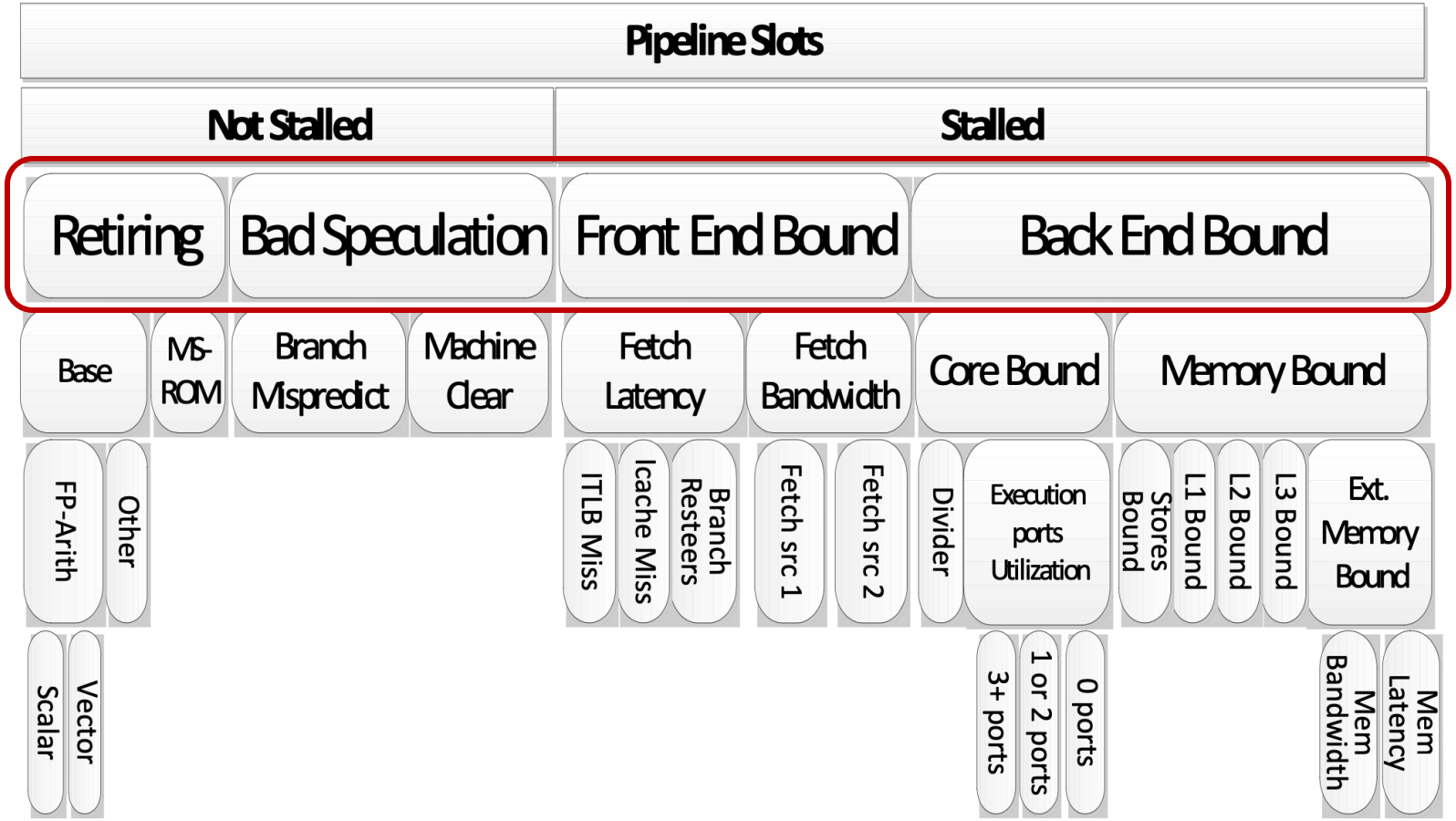
Successfully delivered uops who eventually do retire

The front end is not delivering enough uops per cycle while the back end of the pipeline is ready to accept uops

Tracks uops that never retire or allocation slots wasted due to recovery from branch miss-prediction or clears

No uops are delivered due to lack of required resources at the back end of the pipeline

TMAM: low level breakdown



Supercomputers

#9*Marconi100 – IBM Power9 + Nvidia V100

#47* Marconi – Intel Xeon 8160 SkylakeX

#7* Selene – AMD 7742 Epyc + Nvidia A100 (DGX)

#203* Astra – ARM Cavium ThunderX2 (ARMIDA@E4)

*at ranking time

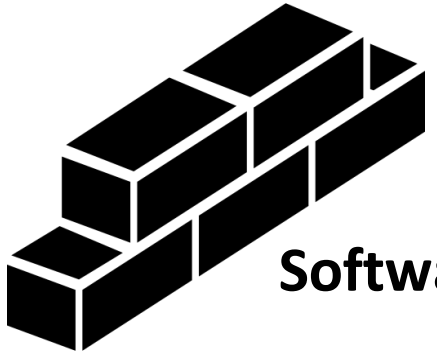
Why did we select these supercomputers?

CPU	ISA	Cores	SMT	Operating Frequency (GHz)	Peak IPC	Peak FLOPS (GFLOPS)	Peak Memory Bandwidth (GB/s)	TDP (W)	Peak Energy Efficiency (GFLOPS/W)	Characteristic
Intel Xeon 8160	x86-64	24	OFF	2.1 (AVX-512)	4 (6)	1612	120	150	10.75	Unbalance on the compute
IBM Power 9	Power ISA	16	x4	3.8 (VSX-128)	6	486	160	250	1.94	Unbalance on the memory
Cavium ThunderX2	ARMv8.1	32	OFF	2.5 (NEON-128)	4	640	160	200	3.20	Unbalance on the memory
AMD 7742 Epyc	x86-64	64	OFF	2.25 (AVX-256)	4 (8)	2662	190	225	11.82	Good overall balance

*nodes of all the systems are double sockets

Applications & Software Stack

MATERIALS DESIGN AT THE EXASCALE
EUROPEAN CENTRE OF EXCELLENCE



Software Stack

QUANTUMESPRESSO

www.flapw.de
fleur

BigDFT

CP2K

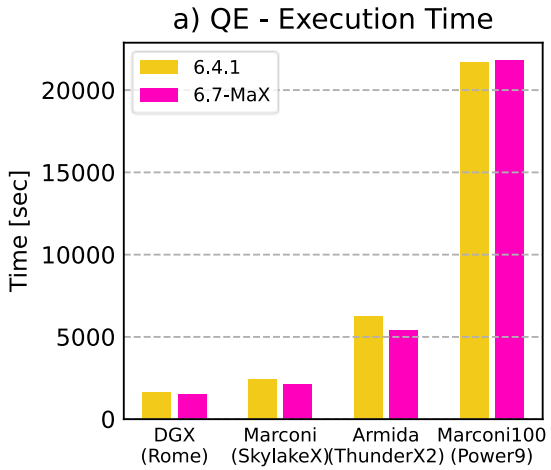
Yambo

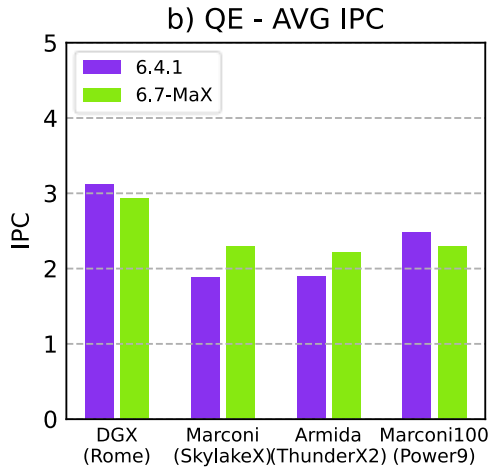
siesta™

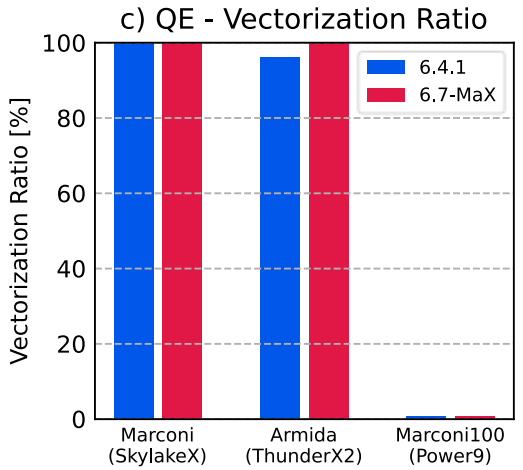
CPU	Compilator	MPI	BLAS & LAPACK	ScaLAPACK	FFTW
Intel Xeon 8160	GCC 9.3	OpenMPI 4.1.1	MKL 2018	MKL 2018	MKL 2018
IBM Power 9	GCC 9.3	OpenMPI 4.1.1	ESSL 6.2.1 OpenBLAS 0.3.18	Netlib-scalapack 2.1.0	FFTW 3.9
Cavium ThunderX2	GCC 9.3	OpenMPI 4.1.1	ArmPL 20.3	Netlib-ScaLaPACK2.1.0	FFTW 3.9
AMD 7742 Epyc	GCC 9.3	OpenMPI 4.1.1	AMDblis/Flame 3.0	AMDScaLaPACK 3.0	AMDFFTW 3.0

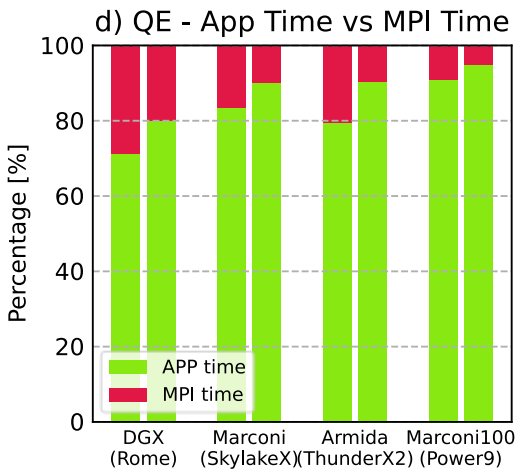
Other accessory libraries:

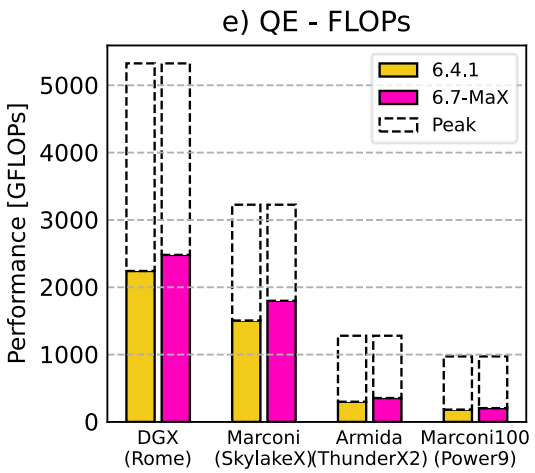
- Netcdf Fortran 4.5.3
- Netcdf C 4.8.1
- HDF5 1.10.7
- ELSI 2.7.1

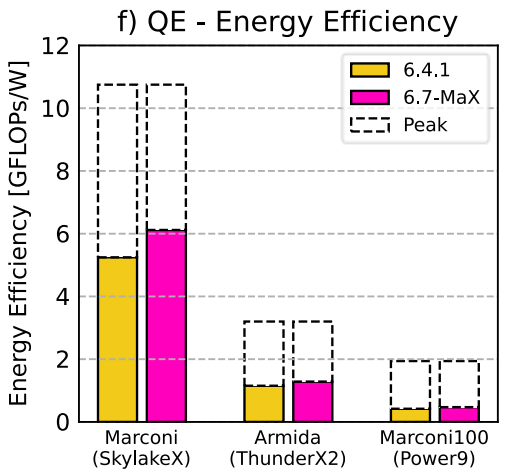












Configuration:

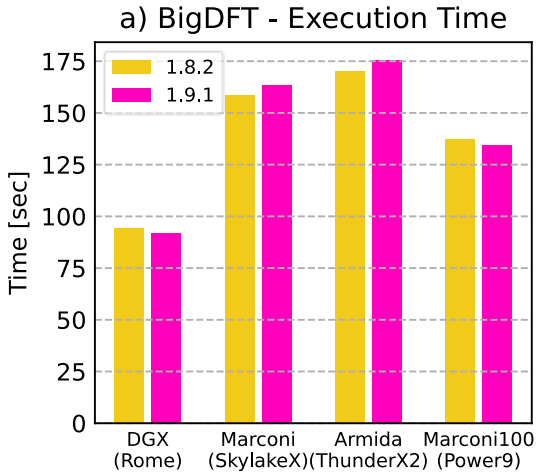
- Single node experiments
- Pure MPI (one MPI rank per core)
- We compared two different version of the code

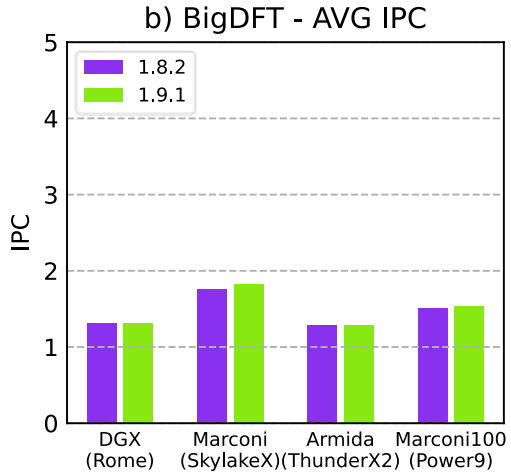
Behaviors:

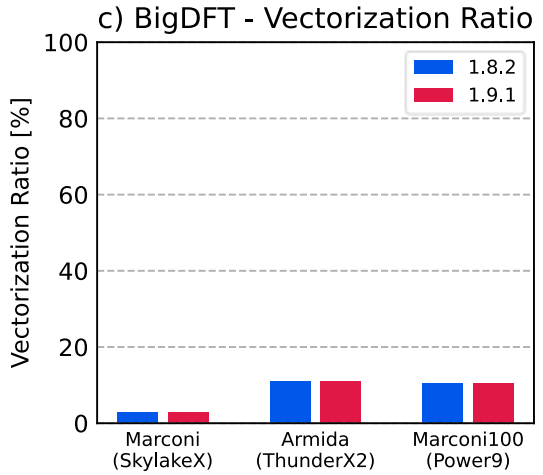
- *Execution time*: proportional with the peak FLOPS (Power9?)
- *IPC*: ≥2 limited by the number of vector ALUs (2)
- *Vectorization ratio*: ≥100% (Power9?)
- FLOPS and FLOPS/W: proportional with the peak FLOPS

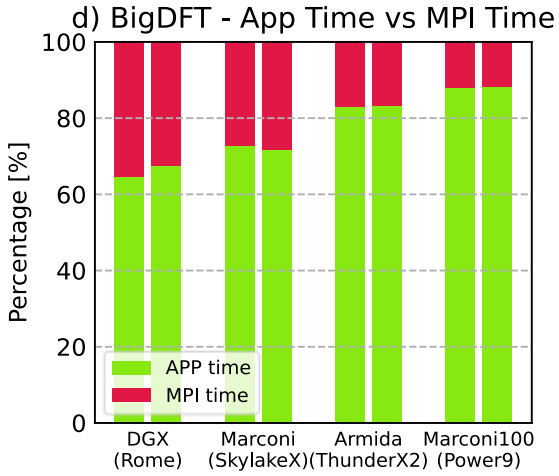
Conclusions:

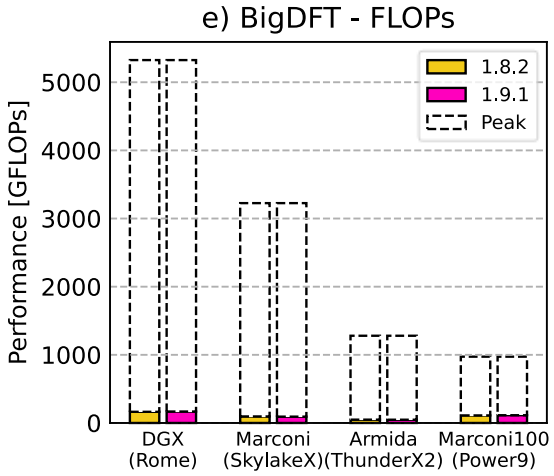
- QE is compute bound!
- High vectorized codes work pretty well!
- Intel µarchs ❤ QE

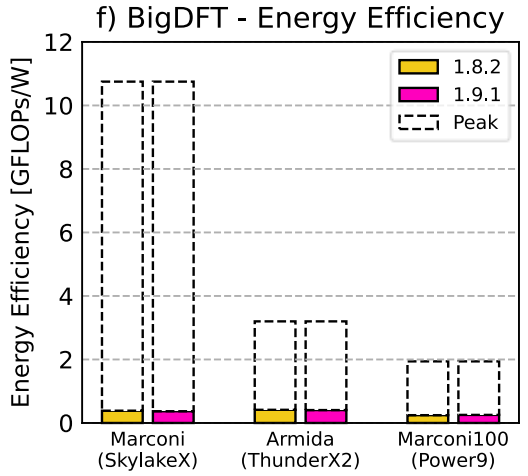












Configuration:

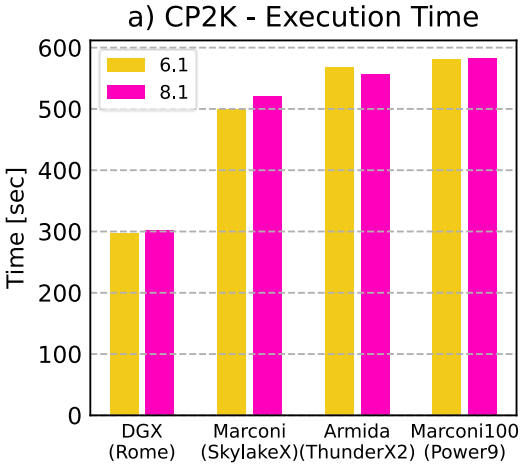
- Single node experiments
- MPI & Openmp
- We compared two different version of the codes

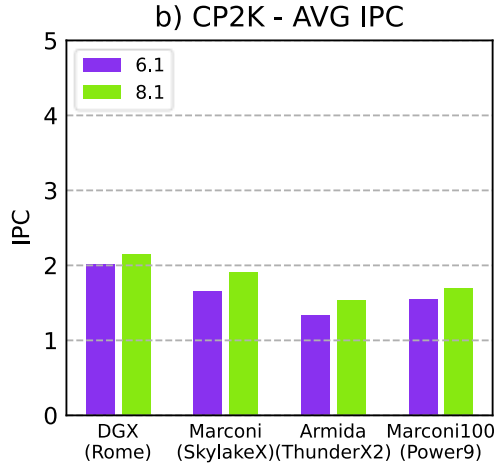
Behaviors:

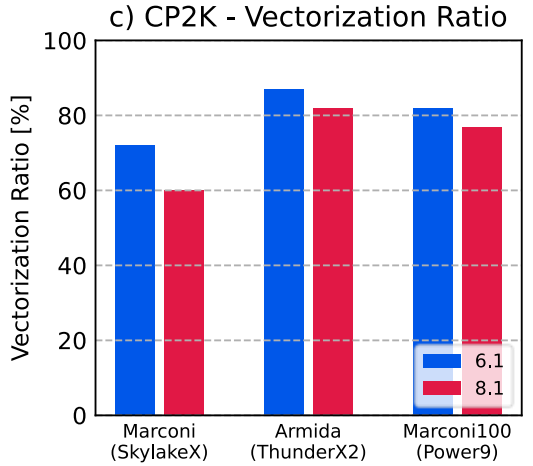
- *Execution time*: proportional with the peak memory bandwidth and the capacity to move data!
- *IPC*: <2 limited by the memory
- *Vectorization ratio*: <15% this workload is very difficult to vectorize!
- FLOPS and FLOPS/W: limited by the memory, are these good metrics?

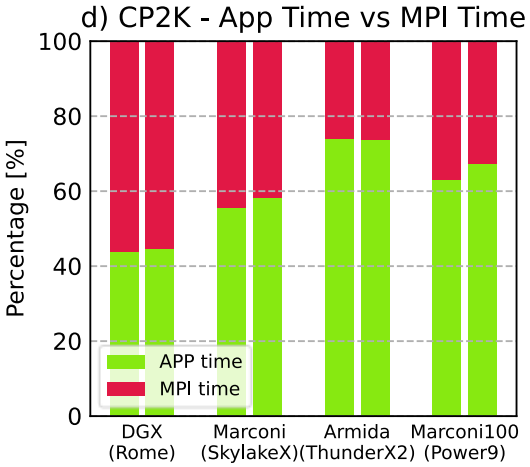
Conclusions:

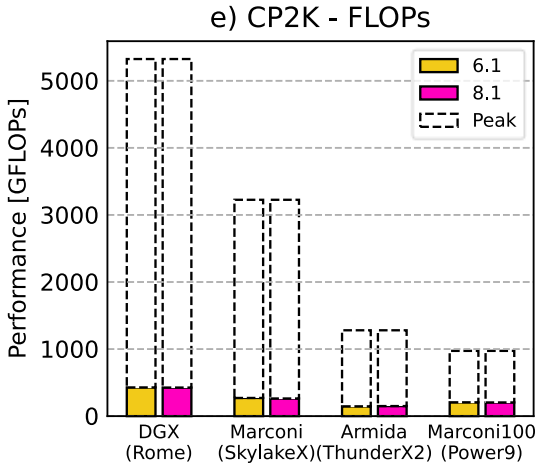
- BigDFT is memory bound!
- IBM Power9 ❤ BigDFT

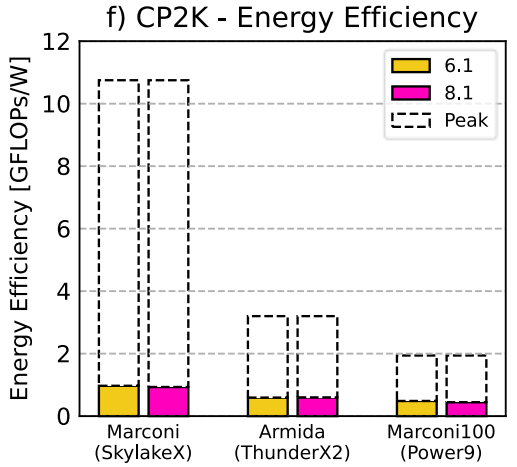












Configuration:

- Single node experiments
- Pure MPI (one MPI rank per core)
- We compared two different version of the codes

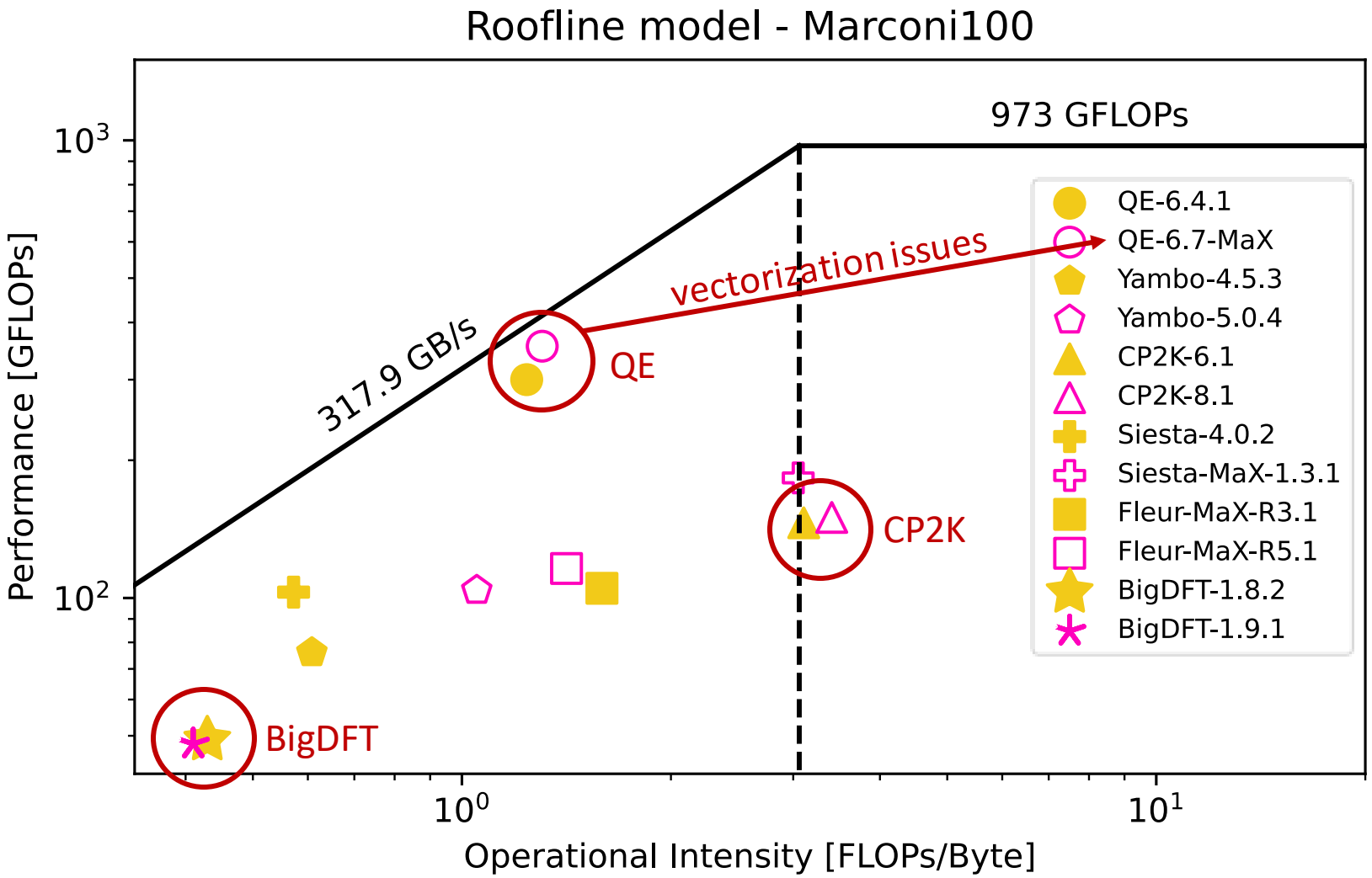
Behaviors:

- *Execution time*: very similar for Skylake, ThunderX2 and Power9
- *IPC*: ≤2 compute needs memory (and vice versa)
- *Vectorization ratio*: >60%, it's not a vectorization problem!
- *APP time vs MPI time*: very high, CP2K needs a lot of communication -> memory!
- *FLOPS and FLOPS/W*: very low but they are limited from the memory and communication

Conclusions:

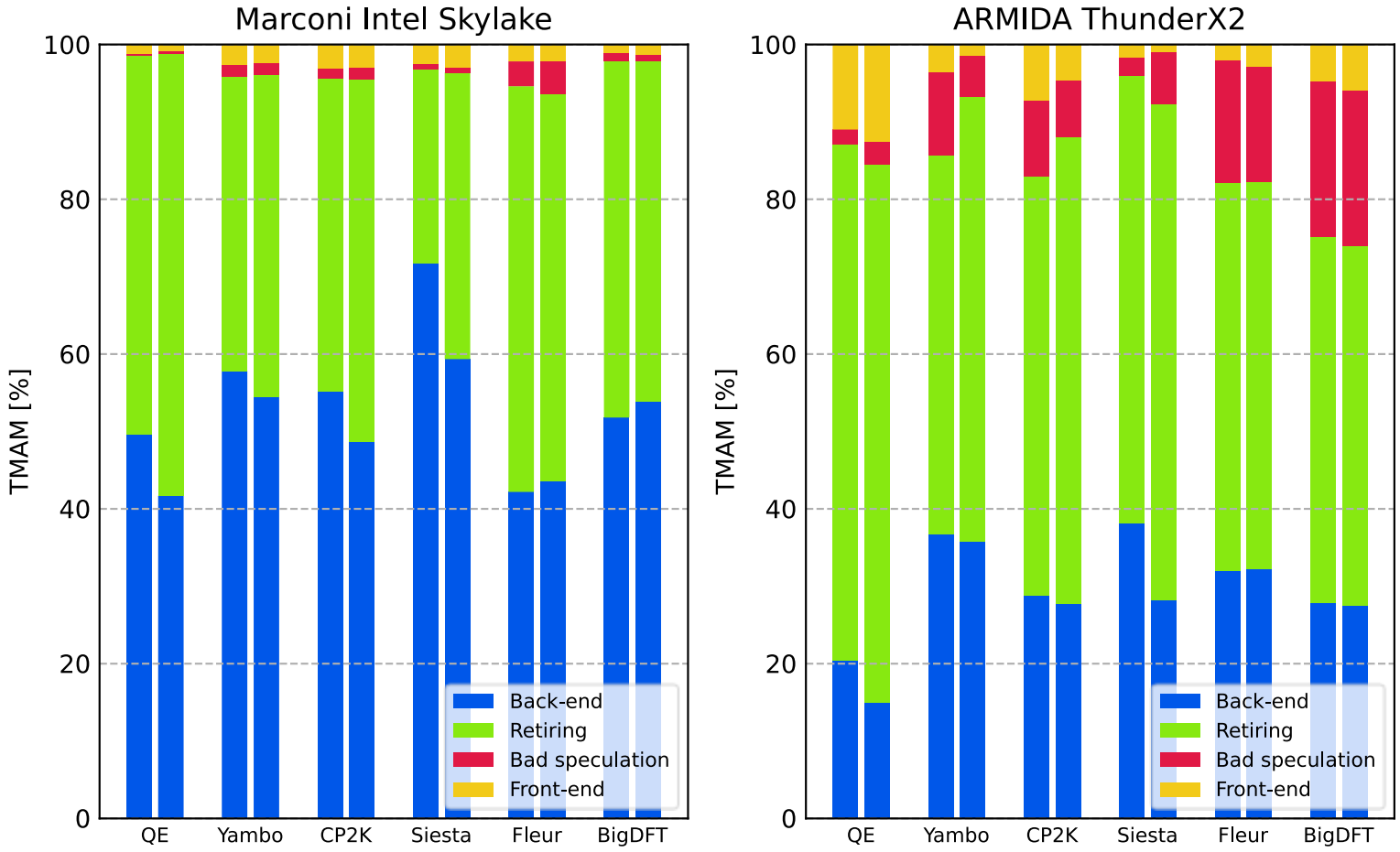
- Fleur is a balanced application!
- CPU with a good balance between compute and memory are the most efficient!
- AMD ❤ Fleur

Roofline – IBM Power9



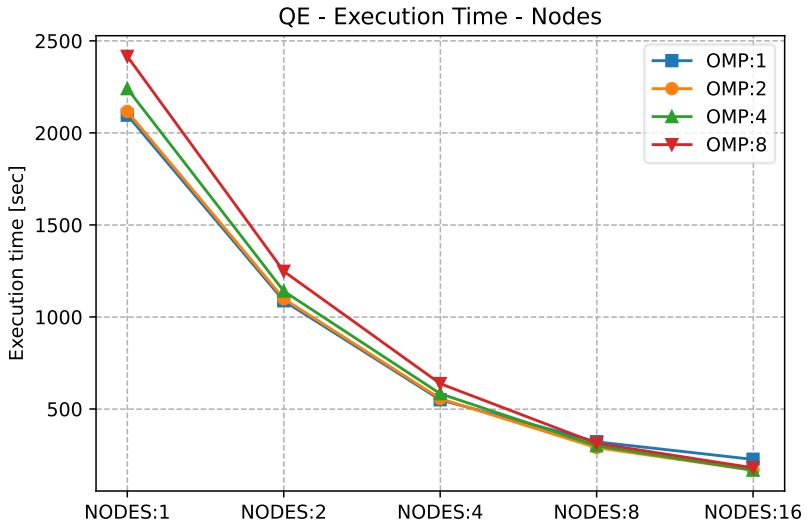
*single node experiment results

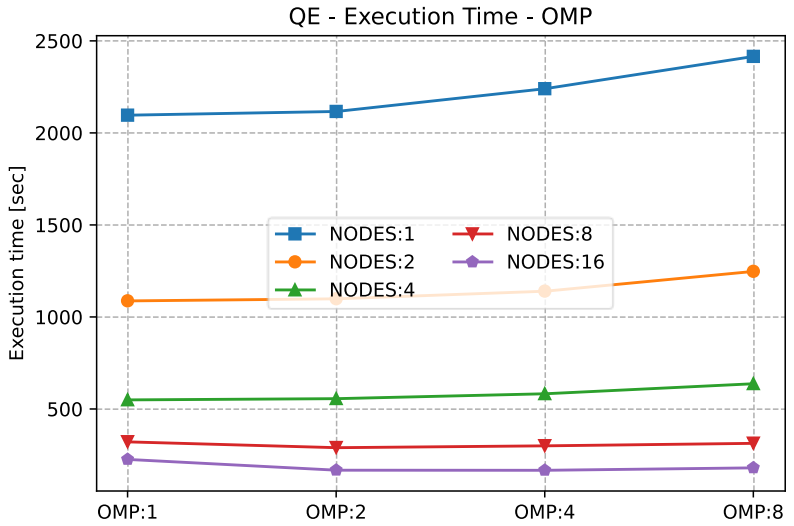
TMAM – Intel SkylakeX vs ARM ThunderX2

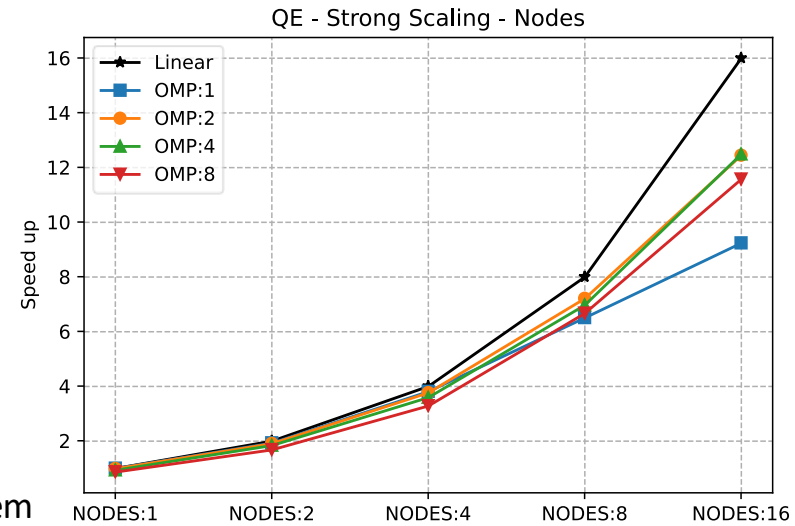


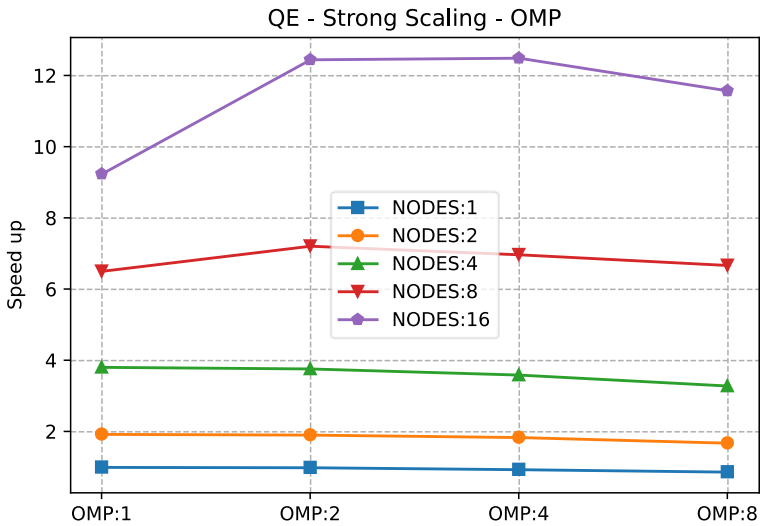
*single node experiment results

Performance and power efficiency at scale with MPI and OpenMP



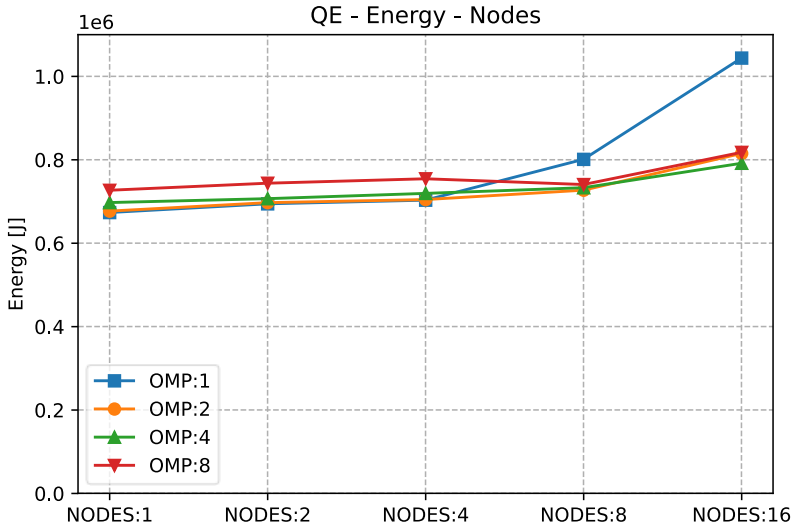


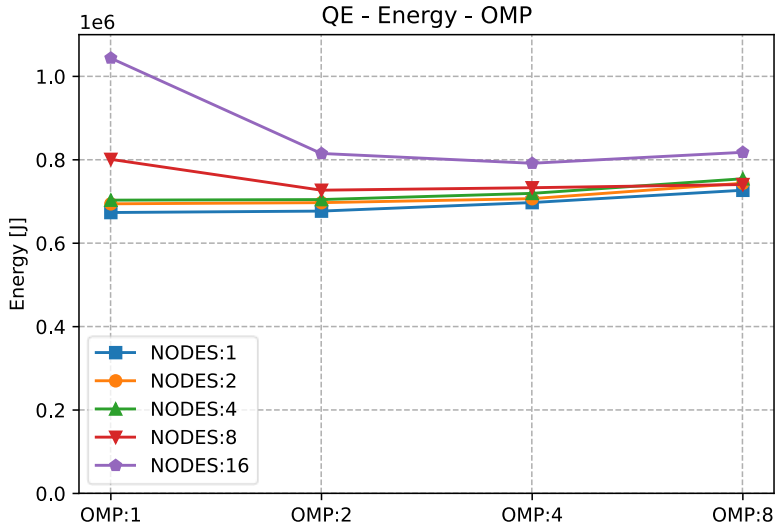


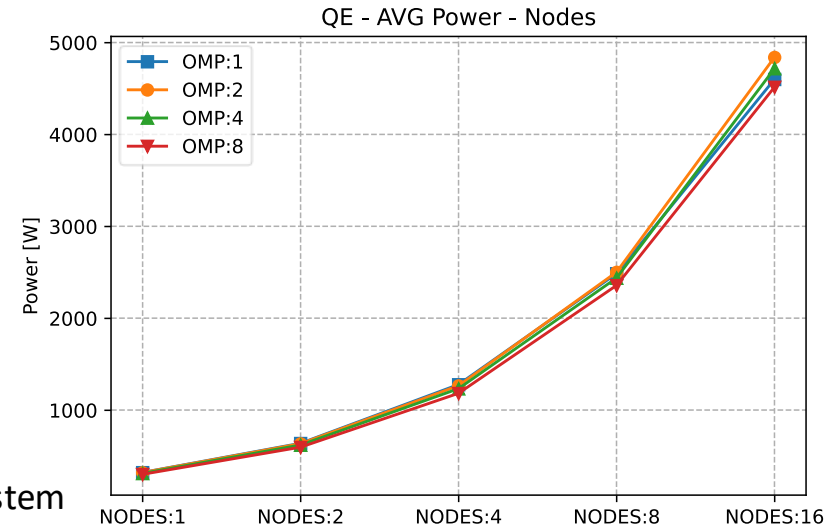


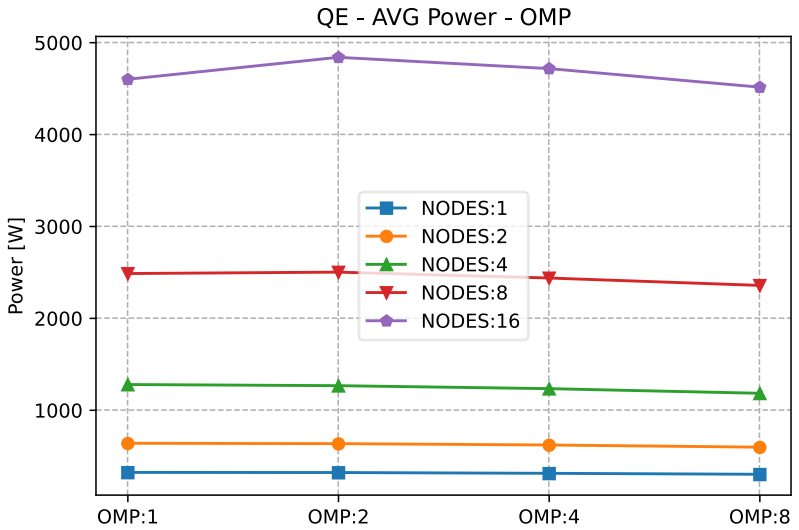
*Marconi HPC system

Performance and power efficiency at scale with MPI and OpenMP









*Marconi HPC system

Conclusions

- *MaX codes have very different performance on different architectures*
- *MaX codes are typical HPC applications -> pretty memory bounded!*
- *Only few of them are compute bound, but only if they can leverage on optimized HW accelerated libraries*
- *Complex codes need complex microarchitecture to perform efficiently*