SISSA - ICTP

MASTER IN HIGH PERFORMANCE COMPUTING

Master's Thesis

APPLICATION-LEVEL ENERGY PROFILING: THE CO.S.IN.T. CASE STUDY

Author:
Moreno Baricevic

Supervisors:
Stefano Cozzini
Andrea Bartolini

ACADEMIC YEAR 2014-2015

Abstract. This thesis aims to explore feasible methods to profile, from the energy efficiency point of view, scientific applications by means of the performance counters and the model-specific registers provided by Intel family microprocessors. The Aurora HPC system[1] used as testbed was made available by courtesy of CO.S.IN.T.[2], and it belongs to the same family of machines that enabled the Eurora[3] cluster hosted at Cineca[4] to reach the 1st rank in the 2013 Green500[5]. Retrieving power consumption information through NVIDIA accelerators and external monitoring and management devices on such platform is explored too.

Keywords: application-level energy profiling, energy efficiency, RAPL, top-down characterization, performance counters, HPL, BLAS comparison, HPCG

Table of Contents

1. Introduction

In this section, the main concepts and the motivation behind this work will be introduced.

1.1. The energy problem in HPC

Energy-consumption is one of the main limiting factors for the development of High Performance Computing (HPC[1]) systems into the so-called "exascale" computing, which consists of systems capable of at least one exaFLOPS. Indeed, the Top500 project[6], which ranks and details the 500 most powerful computer systems in the world, already reports power consumption in the order of the thousands kilo Watts for systems with “few” petaFLOPS of computational power. As of 2015/11, 212 supercomputers declare a total power of over 1 MW, 3 of which over 10 MW: "Tianhe-2"[7], the supercomputer currently ranked at the top of the list with 34 PFLOPS, requires 17.8 MW, and "QUARTETTO"[8], a cluster ranked “only” 67th with 1 PFLOPS, requires even more, 19.4 MW. Following this trend, an exascale supercomputer built with the technology available today would demand a power budget in the order of 500 MW, sustainable only with a dedicated power plant. For this reason feasible exascale supercomputers will have to fulfill the 20 MW power budget.[9] Reaching this target will require an energy-efficiency of 50 GFLOPS/W, one order of magnitude higher than today's greenest supercomputer. In fact, the Green500[10] list, which sorts the Top500 by energy-efficiency, ranks Tianhe-2 only 90th and QUARTETTO 499th. The greenest computer as of 2015/11, “Shoubu”[11], is capable to deliver 7 GFLOPS/W (ranked 135th in the Top500), which would still need 143 MW if scaled up to exascale.
Several power management strategies can increase dynamically the final energy-efficiency of a supercomputer by keeping the power consumption under control or by assessing the energy efficiency online. Practical examples can span from the power capping features of a job scheduler, to the (unattended) energy profiling of an application, tasks taken into exam with the project hereby presented.

In addition, due to energy cost and availability, power consumption of data centers is facing as one of the rising problems, thus energy efficiency is rapidly becoming a hot topic, especially moving towards exascale systems. "Green" computing is now often coupled to high-performance computing, introducing a new concept of "high-efficiency" computing, especially due to the fact that the total cost of ownership (TCO) of a HPC infrastructure is largely impacted by the power consumption during its life cycle.

This growing interest in energy efficiency is leading to new approaches to HPC as well as new hardware requirements. Sysadmins, engineers, decision makers (and eventually users) are becoming aware of the energy problem. There's a growing need for energy-aware resource management and scheduling, in order to implement and enforce power capping constraints. Doing this, though, requires probes and tools for monitoring the power consumption of a system and change the power consumption of the hardware at run-time. To accomplish this, hardware/software sensors report near-real-time power consumption. The software interfaces for accessing this information at run-time are rapidly evolving and more often integrated in high-end infrastructures.

1 Many acronyms will be used throughout this document. Their explanation and expansion can be found in Appendix A.

At the monitoring level, novel concepts are emerging to increase the final user awareness on the energy efficiency. Energy-to-solution can be used to account the energy dissipated by the execution of the user code while energy profiling of the application provides more details about the "cost in energy" of running a specific code on a specific machine, and using a specific library.

At the power management level, several methods can be taken into account in order to reduce the power consumption of HPC resources by adapting their performance level to the workload demand.
Power wasted by idle resources can be reduced by mean of software power management policy, which will automatically put the idle resource into power saving modes (sleep, standby, power-off), and will power on/wake up the nodes when new workload is available. PDU power off can be used to further reduce the total power consumption.
Some hardware capabilities can be exploited too, for instance the Advanced Configuration and Power Interface (ACPI) defines sleeping states (S-states), power states (C-states) and performance states (P-states, for instance by means of Dynamic Voltage and Frequency Scaling (DVFS), Turbo Boost technology, RFTS mechanisms and power/clock gating) in order to dynamically configure and monitor the power consumption. All these features, which are implemented at hardware level by the CPUs, can be enabled by compliant motherboard's BIOS and exposed as a control knob to the operating system for run-time power-optimization.
While dynamic power management approaches, which trade-off performance for energy efficiency, may affect the final user QoS on the supercomputer infrastructure, emerging power capping policies [12] allows to constraint the total power consumption of the supercomputer by admitting only the jobs which fulfill a required power budget. A run-time-enforced reduced power budget saves energy by avoiding cooling over-provisioning. In addition by selecting the highest performance-per-watt resources first the overall energy-efficiency can be improved.
All the above strategies are now granting a lot of attentions, and a significant effort from the HPC industry and research community is focused on the development of energy-aware resource management systems and schedulers.

All these techniques, to be effective, require run-time access to the system and power consumption status. Recent generations of microprocessors introduced specific registers that allow to measure the power consumption of different sub-systems (logical and physical areas) of the CPU with fine granularity. For the Intel family of processors, these metrics can be obtained accessing specific hardware counters through the Model-Specific Register (MSR) interface, in particular the so-called Running Average Power Limit (RAPL), which allows to monitor, control, and get notifications on System-on-Chip (SoC) power consumption (platform level power capping, monitoring and thermal management). RAPL is not an analog power meter, but rather estimates current energy usage based on a model driven by hardware performance counters, temperature and leakage models, and makes this information accessible through a set of counters[13].

In addition, large infrastructures can count on several sensors and devices, for instance local machine hardware sensors, power distribution units (PDU), power grid counters, external hardware dependent probes and sensors. High-end machines offers out-of-band management and monitoring capabilities (independently of the host system's CPU, firmware, OS), through the so-called Intelligent Platform Management Interface (IPMI), which allows the monitoring of system temperatures, voltages, fans, and power supplies.

Other external devices can be instead accessed via various standard network-based protocols, like the Simple Network Management Protocol (SNMP), or proprietary protocols and connections.

Additional local hardware probes and sensors interfaced to the motherboard (depending on the vendor and models) can be queried using `lm-sensors` (Linux monitoring sensors), which provides tools and drivers for monitoring temperatures, voltage, humidity and fans. The `fam15h_power` driver, for instance, permits to read the CPU registers providing power information of AMD Family 15h processors, similarly to what RAPL does for Intel microprocessors, although without the same level of granularity.

1.2. Motivation and approach

Power capping capabilities, energy-based policies and energy-aware scheduling, all require some insights concerning the power consumption of the system. Choices concerning the evolution of such systems must be based on the ability to estimate the power consumption of a specific job depending on the resources requested and the kind of task or calculation that will be performed. CPU-bound application will draw more power from the CPU, and probably less from memory and storage. Memory-bound applications will demand more activity from memory and storage than from the CPU, thus moving the larger power consumption to other sub-systems, while the CPU will be probably idling most of the time. Real world applications are often a combination of both, and identifying which part will prevail may be an indication of what the power consumption could be.
Therefore, the capability of collecting application-based energy profiles and monitoring the power consumption of the entire system, will allow an energy-aware scheduler to predict the trend based on the applications that are queued for execution, and hence schedule the jobs in a way that allow to respect imposed power capping constraints.

The purpose of the first phase of this project was to determine an energetic profile for some well-known scientific applications, by identifying, in particular, the power consumption relative to some specific routines or hardware activities. Furthermore, this profiling included the comparison between different BLAS implementations, as supplied by widely used mathematical libraries, for the same application, thus highlighting how different implementations of the same family of routines can affect the power consumption in order to solve the very same problem.

Most of the tests in this phase involved the High Performance Linpack (HPL) and the High Performance Conjugate Gradient (HPCG) benchmarks. Besides being well-known applications, used to characterize and rank the clusters of the Top500, these tools were chosen because the first is a CPU-bound application, while the second is memory-bound, difference that allowed to perform a study of the trade-off between energy consumption and performance by changing the frequency of the CPUs (DVFS).
HPL was compiled in several versions making possible to compare the performance delivered by various BLAS implementations, in particular Netlib[14], ATLAS[15], OpenBLAS[16], PLASMA[17], Intel MKL[18].

In the second phase of this project, a couple of real-world scientific applications was tested, Quantum ESPRESSO[19] and LAMMPS[20]. All the methods used to analyze the previous benchmarks were then used on these applications. A lot of efforts was dedicated to the analysis of the performance counters and the top-down characterization, issues explained in details in section 2.3 and throughout section 3.

With the exception of Intel's MKL, only Free and Open-Source Software (FOSS) was employed in this study. NVidia's CUDA accelerated Linpack was also used in this research, even though not strictly open.

The outline of this work is as follows: in section 2 the testbed will be explained both at hardware and software level, including the technological issues and solutions explored and eventually adopted, as well as the way the benchmarks were chosen. In section 3, the results obtained will be exposed and discussed in details, for each type of analysis performed throughout this work. Finally, section 4 is devoted to the conclusions and the future perspectives that this research stimulated.

A rich set of appendices complete the work. Some specific details concerning analysis methods are reported in Appendix B. and Appendix C., while the results of exploratory testing are presented in Appendix D. Finally, many of the plots produced and cited in this document have been collected in Appendix E. and Appendix F.

2. Testbed

In this section, the development platform will be introduced and detailed. In particular, the hardware, the software and the technological peculiarities of the approach are explained in dedicated subsections.

2.1. Hardware

The basic requirements to perform this work were:

- super-user's privileges;
- dedicated master node + at least 2 computing nodes;
- Intel family processors supporting RAPL.

The infrastructure in production at COSINT (Amaro, UD, Italy) was able to fulfill all the requirements, and offered even more. The available platform, an Aurora[1] system developed by Eurotech S.p.A., was an ideal platform for this kind of analysis since it belongs to the same architecture that reached the 1st position in the 2013 Green500[5], the “greenest” platform, with 3.2 GFLOPS/W.

Full control over a virtual machine (VM), used for testing `SLURM`[21], and a computing node was initially granted full-time for the whole period covered by this project. Few more computing nodes were made available occasionally for scaling tests, and external monitoring devices were made accessible too.
The Aurora computing nodes under exam are equipped with 2 Intel Xeon Ivy Bridge processors with 12 cores each at 2.7 GHz, 64 GB of RAM, and 2 NVidia K20 GPUs.

In detail, the tests on this platform involved the following machines:

- 1x masternode / access node;
- 1x AURORA chassis with 6 blade (no accelerators);
- 1x AURORA chassis with 4 blade (2x NVIDIA GPU K20 on each blade);
- 10x blades with 2x 12-cores CPU Intel Ivy Bridge @2.7 GHz, and 64 GB of RAM;
- 1x virtual machine installed and configured as `SLURM` server;
- 2x blades configured as `SLURM` compute nodes.

The chassis used for most of the tests may be reported in plots and tables as "chassis 2".
The blade used for most of the tests is "b21" ('b'lade node, chassis #2, blade #1).

Unfortunately, the original design of the Aurora platform, developed with the clear intent of providing an innovative and energy-friendly system (successfully), presents some side effects too. Due to the peculiar experimental nature of the platform (in between a technology demonstration and a production-ready architecture), some tools and sensors often available and used on mass-produced

high-end platforms are not available or just not implemented yet. For instance, the IPMI interface that manages the blades cannot be queried in order to obtain temperature and power readings, and the system does not offer any alternative interface or remotely-accessible sensor for this purpose.

Nevertheless, the external power supply can be queried via SNMP and it is able to provide real-time voltage, current, and temperature readings from the monitoring sensors of its 6 power rectifiers (electrical devices that convert alternating current (AC) to direct current (DC)). Each of the 2 available chassis of blades are directly powered by 3 of these 6 rectifiers. By combining the data collected from the 3 rectifiers connected upstream, it is possible to derive the total power absorbed by *all* the blades of a chassis, although this value includes the chassis itself (rootcard controller, fans, ...). The granularity and precision are clearly suboptimal, but the idea was to obtain differential readings in order to exclude the background power consumption of the chassis and idling blades.
In order to obtain as much data as possible for the study of the power consumption, some tests hence included the reading obtained querying these devices while the benchmarks were running on one or more nodes at the same time (while the other nodes were idling). The analysis of these readings is reported in Appendix D.1.

Most of the analysis hereby presented are based upon the microprocessor and its features. The processor in use on the test platform is the Intel Xeon Ivy Bridge EP E5-2697 v2 @ 2.70 GHz[22], with 30 MB of L3 cache, in a dual-sockets server platform configuration (code-named Romley).

Section 2.3 will reveal additional details concerning the features of this family of processors.
The Figure 2.1:1 shows the internal topology of the processor as reported by `lstopo` (`hwloc`[23]).

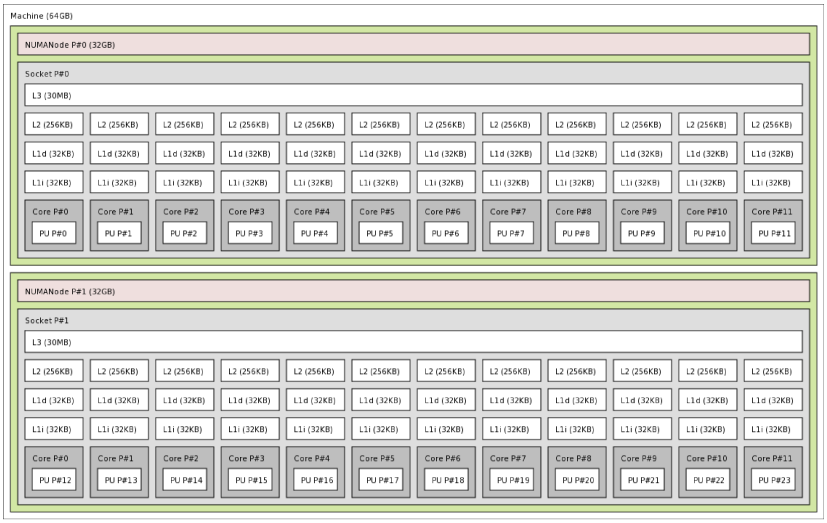


Figure 2.1:1: CPU internal Topology (`lstopo`/`hwloc`)

2.2. Software

This section provides all the details concerning the software layers of the test environment. The needs, and the software tested, developed, and eventually adopted will be discussed.

2.2.1. General overview

The operating system used is Linux, in particular a CentOS distribution [24], version 6.5, with the stock kernel 2.6.32-358.23.2.el6.x86_64 and default utilities. Additional packages, detailed in sections 2.2.2 and 2.3, have been installed to perform low-level operations, like handling the power governor, the frequency scaling, and the performance counters.

The libraries and executables were compiled using the GNU compiler version 4.8.3 [25] and OpenMPI version 1.8.3 [26]. A license for Intel compilers and MPI was not available on the platform used. Any difference in performance, though, was deemed irrelevant for the kind of analysis meant to be performed on the system and on the software stack.

Even though the cluster was configured to use `PBS-Pro`[27] as resource manager and job scheduler, it was not involved during most of the tests, as direct access to the computing nodes was preferred (avoiding `cpusets` and other unwanted features). `PBS-Pro` doesn't currently support out-of-the-box energy-based scheduling policies, power capping mechanisms and per-job energy reports.
`SLURM`[21], though, an open-source workload manager, is rapidly evolving in this direction, already allowing to enable the report of per-job power consumption, and it is rapidly moving toward implementing and providing power capping capabilities and energy-based fair-sharing too.[28][29]
A virtual machine and a couple of computing nodes were temporary dedicated to setup such framework (`SLURM` 14.11.7), and some tests have been conducted to investigate its features. The per-job energy reports, in particular, were used to compare the results obtained by the means of other monitoring utilities.

The Eurora Monitoring Framework [30], a set of scripts and utilities developed by Micrel Lab[31]/UNIBO[32]/CINECA[4] and used for similar analysis on the Eurora cluster[3], was used in order to access the hardware performance counters and record the power consumption of a node during the run of the application under exam.
In the setup phase, the framework was adapted and improved. Some utilities, in particular the one which acquires the MSR/RAPL counters, called `msr-statd`, and the one devoted to filter the data (`msr-filter`), have been modified in order to work on the COSINT cluster, and virtually, anywhere else too. In particular, these utilities have been interfaced to `hwloc` and `numactl`[33] in order to automatically detect the hardware configuration at runtime (number of sockets, number of cores, core-to-socket map), and many command-line options have been added in order to modify at will all the vital parameters, hardcoded at compile-time in the original version.
The overhead of this utility was considered negligible as its CPU-time was measured in 1.2 milliseconds for each reading performed at regular intervals of 10 or 5 seconds. Should be noted that, due to the size of the registers and the frequency these registers are updated by the CPU, the interval must be kept reasonably short (<30 s), as some counters regularly overflows and the software must be able to notice this and handle it to compensate.

The modified code was then compared to the energy readings obtained through `SLURM` and `LIKWID[34]`. From a first analysis, by detecting the power consumption of a run of HPL, the values

reported by SLURM and msr-statd resulted to be compatible and reliable, likwid-powermeter, though, reported values that wasn't possible to associate clearly to specific resources utilization. By using LIKWID versions 3.1.3 and 4.0.0, the values reported were not even compatible with each other. After these results, LIKWID was not used for further testing. Should be noted, though, that at the time of this writing, the current git version of LIKWID (as retrieved the 2015/10/16) reports values compatible with the readings obtained with SLURM and the msr-statd utility.
At the end of this comparison, the software developed by Micrel Lab was chosen as it suited best all the requirements, as it is stand-alone, out-of-band (no instrumentation of the tested code is needed), flexible, easy to use and to adapt.

Performance monitoring events represent a powerful tool for the profiling and improvement of the performance of an application. Some of these performance events permit to understand whether an application is memory-bound or CPU-bound, or maybe bound to other components such as the GPU. This allows to identify where the application is bottlenecked and possibly indicate how to improve it.
A preliminary phase was devoted to study the available Linux utilities, libraries and APIs for the management and monitoring of system performance and energy consumption, as well as programming techniques for low-level access to CPU performance counters and registers.
The various tools and techniques that have been used to retrieve and analyze such kind of information will be illustrated and discussed throughout this document.

Several ad-hoc scripts and utilities were developed in order to collect other pieces of information and useful data for the analysis. Some of these scripts were used to access, for instance, the power rectifiers via SNMP, while many others were just wrapper scripts for "perf" and other utilities (msr-statd, cpufreq, ...), in order to automate the simulations by varying many of the parameters (core frequency, problem size, performance counters, ...). Many others were written and used to collect, parse and filter the huge amount and variety of data obtained. All these tools have been collected in a git repository and will be soon published as open-source software.

2.2.2. Power management and acquisition software

Various software was investigated, tested and used in order to acquire and collect the information needed, as well as for managing the environment. This section briefly reviews them, giving also some usage examples.

As mentioned in the previous section, the software developed by Micrel Lab/UNIBO on their study of the Eurora cluster, was modified and widely used to obtain RAPL power readings from the CPU.
msr-statd is a MSR/RAPL acquisition software written in C, linked to hwloc and numactl, that requires super-user's privileges in order to access the MSR kernel module interface (/dev/cpu/*/msr).
Usage example:

```
msr-statd --hwloc --background --path $PWD --prefix xhpl.openblas --interval 5 --truncate
msr-filter --input xhpl.openblas.msr.log > xhpl.openblas.msr.log.report
msr-compact.pl < xhpl.openblas.msr.log.report > xhpl.openblas.msr.log.summary
```

Concerning GPU power readings, a python script named gpu-statd (as well part of the Micrel Lab's Eurora Monitoring framework) interfaced to the NVidia Management Library (NVML)[35] provided power consumption and load of the GPUs. Similar readings, even though with a far larger overhead, can be obtained by using the utility nvidia-smi[36].
Usage example:

```
gpu-statd start --path $PWD --prefix xhpl.nvidia --ts 5
gpu-filter.py --input xhpl.nvidia.gpu.log --gpus 2 > xhpl.nvidia.gpu.log.report
```

Additional power consumption information was supposed to be collected by accessing the rectifiers and the power management of the Aurora blades. SNMP utils[37] and IPMI tools[38] were used in order to access the external power rectifiers and the IPMI management interface of the blades.
Some ad-hoc utilities were written in order to setup and poll the power supply and get readings at regular intervals.
Usage example:

```
ipmitool -H sp-b21 -U foo -P bar power status
ipmiwrap.sh b21 sensor list
snmpwalk -mALL -v2c -cfoobar pdu .1.3.6.1.4.1.10520.2.1.5.6.1.8
snmpbulkwalk -mALL -v2c -cfoobar pdu .1.3.6.1.4.1.10520.2.1.5.6.1.10
pductl -f status all
pductl -f pout 2
```

`cpufreq-utils`[39], a user-space utility provide by the kernel tools, was used to query and modify the frequency scaling governor and the CPU frequency, even though it was wrapped by a script to simplify and modularize the utilities used.
(setting frequency and governor requires super-user's privileges).

```
cpufreq-info
seq 0 23 | xargs -t -i cpufreq-set -r -c {} -g userspace
seq 0 23 | xargs -t -i cpufreq-set -r -c {} -f 2700000
```

Direct access to `sysfs`[40] was exploited too for the same tasks. `Sysfs` is a virtual filesystem on Linux, which provides user-space access to kernel objects, like data structures and their attributes. It is possible to read and write various flags, which will be applied by the kernel to the proper sub-system or device.
(setting frequency and governor requires super-user's privileges).
Usage example:
read and set scaling governor:

```
grep . /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
grep . /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor | sort -tu -k3n,3
echo userspace | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

read and set frequency:

```
grep . /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
grep . /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq | sort -tu -k3n,3
echo 2700000 | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq
```

Beside the Eurora framework, `SLURM` energy plugins (`acct_gather_energy`) and utilities provided by `LIKWID` (`likwid-powermeter` and `likwid-perfctr`) were tested to evaluate their capabilities of reporting energy consumption of a job/process, A comparison was performed to verify the power readings obtained and their reliability, in order to figure out the most complete and flexible alternative (`msr-statd` was eventually chosen).
(may require super-user's privileges for some metrics)
Usage example:

```
likwid-powermeter
likwid-perfctr -f -c 0-23 -C 0-23 -g ENERGY mpirun --np 24 xhpl
likwid-perfctr -f -c 1 -C 1 -g BRANCH /bin/ls
```

Linux Perf[41] was extensively used in order to collect the performance counters, needed also to perform the top-down characterization. `Perf` is a native Linux utility that interfaces its kernel-space layer (`perf_events`) to the user-space, allowing to access, read and collect the performance counters during the run time of a process. `Perf_events` interacts with the model-specific registers (MSR) and the performance monitoring unit (PMU) of the CPUs through the `msr` kernel module. The data collected can be post-processed in order to perform a deeper analysis and extract derived metrics, obtaining low-level runtime information about the software under exam, its performance, and its bottlenecks.
(may require super-user's privileges for some metrics)
Usage example:

```
perf stat sleep 1
perf stat -e branch-instructions,branch-misses /bin/ls
```

```
perf stat -o ./perf.log -x, -e r03c,r19c,r2c2,r10e,r30d /bin/ls
perf stat -a -x, -o ./perf.log \
       -e cpu/config=0x003C,name=CPU_CLOCK_UNHALTED_THREAD_P/ \
       -e cpu/config=0x019C,name=IDQ_UOPS_NOT_DELIVERED_CORE/ \
       -e cpu/config=0x02C2,name=UOPS_RETIRED_RETIRE_SLOTS/ \
       -e cpu/config=0x010E,name=UOPS_ISSUED_ANY/ \
       -e cpu/config=0x030D,name=INT_MISC_RECOVERY_CYCLES/ \
       mpirun --np 24 xhpl
```

Finally, a huge number of ad-hoc scripts, filters, parsers and wrappers to run the benchmarks and collect and analyze the data were written in `bash`, `awk`, `sed`, `perl`, `python`, C, `gnuplot`[42].

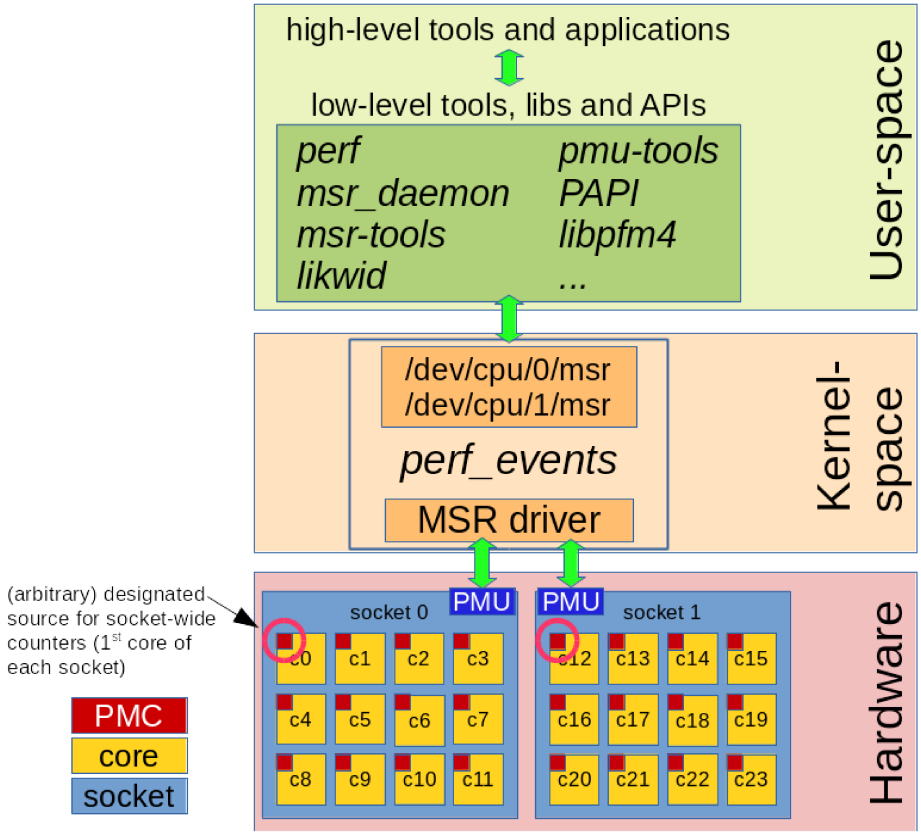


Figure 2.2.2:1: performance analysis SW stack (MSR only)

2.2.3. Benchmarks

Four different applications were used in this work: two HPC standard benchmarks and two scientific applications in materials science. All the four packages are well-known in their respective scientific domain.

High Performance Linpack benchmark and the High Performance Conjugate Gradient benchmark. have been used in the first phase to investigate and calibrate the profiling methods.

High Performance Linpack (HPL) is a portable implementation of the High Performance Computing Linpack Benchmark widely used to benchmark and rank supercomputers for the Top500 list.
HPL is CPU and memory intensive with non-ignorable communication. HPL generates a linear system of equations of order *n* and solves it using LU decomposition with partial row pivoting. It requires installed implementations of MPI and makes use of the Basic Linear Algebra Subprograms (BLAS) libraries for performing basic vector and matrix operations.
The HPL package provides a testing and timing program to quantify the accuracy of the obtained solution as well as the time it took to compute it. The best performance achievable by this software on a system depends on a large variety of factors. The algorithm is scalable in the sense that its parallel efficiency is maintained constant with respect to the per-processor memory usage.[43]

The second benchmark tested is HPCG, the High Performance Conjugate Gradient, a benchmark designed to validate the performance of a supercomputer by simulating an utilization of the resources closer to the real-world applications, often bound to frequent and sparse memory accesses and inter-node communication more than CPU dense computation.
High Performance Conjugate Gradient (HPCG) is a self-contained benchmark that generates and solves a synthetic 3D sparse linear system using a local symmetric Gauss-Seidel preconditioned conjugate gradient method. The HPCG Benchmark project is an effort to create a more relevant metric for ranking HPC systems than the HPL benchmark. Reference implementation is written in C++ with MPI and OpenMP support.[44]
Jack Dongarra, presenting the benchmark[45][46], talks about a “Performance shock”. The performance observed with HPCG can be even less than 1% of the peak performance of a system, far away from those obtained with HPL based mainly on the computing power of the CPU.

The fact that the algorithm is known, the applications are reliable, the input finely configurable, and both provide performance and timing, made of these tools the perfect samples to profile and later use as a reference for comparison. Beside these reasons, HPL is a CPU-bound application, while HPCG is memory-bound. This simple and basic difference permits to figure out how much being dense (or not) in the CPU influences the power consumption of an application.

HPL was tested by linking various BLAS implementations, and for each of them, various performance counters were collected and later used for further analysis.
The power consumption of HPL was also monitored changing frequency scaling governor and forcing different CPU frequencies for each run.
Another test was performed to investigate the hybrid CPU+GPU implementation (optimized and precompiled by NVidia), in order to verify the impact in terms of energy efficiency of moving the computation from the CPU to the GPU and scaling down the CPU frequency.

The tests performed on HPCG include the power consumption by varying CPU frequency, problem size scaling test, as well as performance counters analysis and top-down characterization.

These two benchmarks were largely investigated in order to obtain reliable basis to study additional applications, whose behavior could be unknown, but likely residing within the “extremes” represented by HPL and HPCG.

The next stage was dedicated to the analysis of two scientific applications commonly used in HPC, taken as real-world examples of actual calculations performed on the cluster under exam: Quantum ESPRESSO and LAMMPS.

Quantum ESPRESSO is a software suite for ab-initio quantum chemistry methods of electronic-structure calculation and materials modeling. It is based on Density Functional Theory, plane wave basis sets, and pseudopotentials. The core plane wave DFT functions of QE are provided by the PWscf (Plane-Wave Self-Consistent Field) component, a set of programs for electronic structure calculations within density functional theory and density functional perturbation theory, using plane wave basis sets and pseudopotentials.[19]
The data set used for Quantum ESPRESSO (`pw.x` in particular) is a scaled down input taken from a real simulation performed by a user of the cluster. In this test a Palladium surface is modeled, using a slab geometry. Most of the computational time is therefore spent in linear algebra operations such as matrix-vector multiplications, as well as in fast Fourier transforms.

LAMMPS is a classical molecular dynamics code. LAMMPS has potentials for solid-state materials (metals, semiconductors) and soft matter (biomolecules, polymers) and coarse-grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale. LAMMPS runs on single processors or in parallel using message-passing techniques and a spatial-decomposition of the simulation domain. The code is designed to be easy to modify or extend with new functionality.[20]
The standard Lennard-Jones liquid benchmark, provided with the LAMMPS, was used to profile the application. The input was tuned and scaled up to 70M atoms to make the runtime close to 10 minutes.

2.2.4. BLAS libraries

There are many BLAS implementations available on the market, and each implementation deliver different level of performance. If using a different algorithm to solve the same problem leads to difference performance, it means also that its implementation is able to exploit better (or worse) the underlying hardware. As a consequence, the power consumption, and therefore the energy efficiency, must be impacted too by this different distribution of the resources' utilization.
This study aimed to identify such difference.

This evaluation included the reference BLAS from Netlib, the automatically tuned implementation, ATLAS, the proprietary and closed-source Intel MKL version, and the OpenBLAS highly-optimized free and open-source implementation. The compilation was performed using `gcc` compiler and standard optimization flags have been applied. No further tune was done at compilation phase.

PLASMA[17] was also tested, but it relies on a 3rd party BLAS library, and it can be compiled and linked against any of the aforementioned BLAS implementations. Beside providing its own implementation of many BLAS and LAPACK routines, PLASMA acts simply as a wrapper for many others (in some cases still optimizing their access and utilization). In this case, though, it turned out that the performance achieved using PLASMA was the very same achieved by the underlying BLAS library it was linked to, hence the BLAS routines invoked by HPL were just passed-through. PLASMA was therefore excluded from further analysis.

2.3. Technology

This section provides some technical details concerning the technology surrounding the power management of the microprocessor, the hardware counters and the performance events used and discussed in this work. This section reports also the challenges and difficulties in accessing and interpreting low-level data and the issues faced during the tests and the analysis phase.

2.3.1. Frequency scaling

Microprocessors have seen a continuous evolution during the years, racing to reach higher clock frequencies. The frequency increment, though, led as well to higher power consumption and dissipation. Various solutions have been adopted in order to reduce these factors while maintaining the performance. Aggressive power saving policies have been implemented in order to reduce the power consumption when a resource is not in use or when the highest computing power is not required.

The Advanced Configuration and Power Interface (ACPI) defines sleeping states (S-states), power states (C-states) and performance states (P-states) in order to dynamically configure and monitor the power consumption. All these features are implemented at hardware level by the microprocessors and configurable by compliant motherboard's BIOS and, dynamically, at runtime by the operating system.

Some of these solutions involved methods like the Dynamic Voltage and Frequency Scaling (DVFS), Run Fast Then Stop (RFTS) mechanisms and power/clock gating. Intel processors, in particular, supports Enhanced Intel SpeedStep (EIST) and Turbo Boost technologies, by means of voltage and frequency scaling, internal power capping mechanisms and deep sleep states.

Intel Turbo Boost Technology is a feature that allows the processor to opportunistically and automatically run faster than its rated operating frequency if it is operating below power, temperature, and current limits. The result is increased performance.
The processor's rated frequency assumes that all execution cores are running an application at the thermal design power (TDP). However, under typical operation, not all cores are active. Therefore most applications are consuming less than the TDP at the rated frequency. To take advantage of the available TDP headroom, the active cores can increase their operating frequency.
To determine the highest performance frequency amongst active cores, the processor takes the following into consideration:

- The number of cores operating in the C0 state.
- The estimated current consumption.
- The estimated power consumption.
- The die temperature.

Any of these factors can affect the maximum frequency for a given workload. If the power, current, or thermal limit is reached, the processor will automatically reduce the frequency to stay with its TDP limit.[47]

DVFS is a technique that allows to reduce the power consumption by acting on the frequency at which the CPU is clocked, or its voltage, or both (typically). Lowering the clock frequency, though, usually increases the time-to-solution or walltime (or runtime) of an application. This work aims to verify the influence of this fact on CPU-bound, memory-bound and real-life scientific applications.

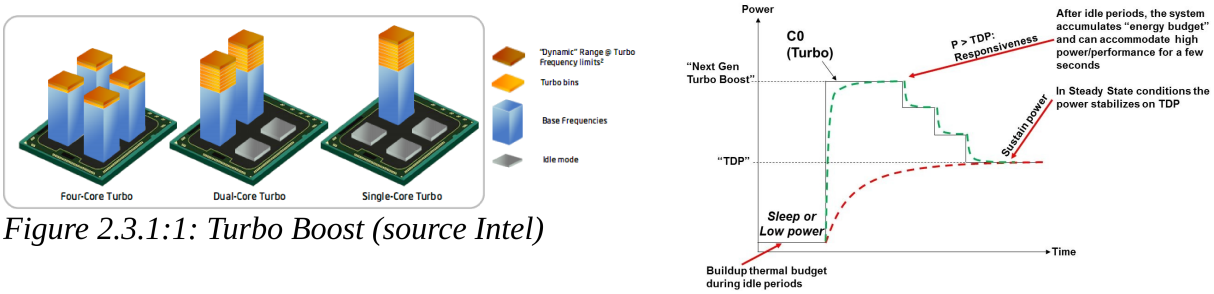


Figure 2.3.1:1: Turbo Boost (source Intel)

CPU frequency scaling enables the operating system to scale the CPU frequency up or down in order to save power. CPU frequencies can be scaled automatically depending on the system load, in response to ACPI events, or manually by user-space programs.

The Dynamic CPU frequency scaling infrastructure implemented in the Linux kernel is called `CPUfreq`[48]. `CPUfreq` is demanded to enforce specific frequency scaling policies, which consist of (configurable) frequency limits (min,max) and `CPUfreq` governor to be used (power schemes for the CPU). Available governors are:

`powersave`**:** sets the CPU statically to the lowest frequency within the borders of `scaling_min_freq` and `scaling_max_freq` (`/sys/devices/system/cpu/cpu*/cpufreq/scaling_{min,max}_freq`)

`performance`: sets the CPU statically to the highest frequency within the borders of `scaling_min_freq` and `scaling_max_freq`.

`ondemand`: dynamically sets the CPU depending on the current usage. Sampling rate (how often the kernel must look at the CPU usage and make decisions on what to do about the frequency) and threshold (average CPU usage between the samplings needed for the kernel to make a decision on whether it should increase the frequency, e.g.: average usage > 95%, CPU frequency needs to be increased) can be defined.

`conservative`: like the "`ondemand`" governor. It differs in behavior in that it gracefully increases and decreases the CPU speed rather than jumping to max speed the moment there is any load on the CPU. Available parameters are similar to "`ondemand`".

`userspace`: allows the (super)user to set the CPU to a specific frequency by making a `sysfs` file "`scaling_setspeed`" available in the CPU-device directory.
(`/sys/devices/system/cpu/cpu*/cpufreq/scaling_setspeed`)

The combination of the running kernel (2.6.32-358.23.2.el6.x86_64) and the IVB-EP processor in use on the test platform allows to select the governors `ondemand`, `userspace` or `performance` (`/sys/devices/system/cpu/cpu9/cpufreq/scaling_available_governors`).

The frequencies that can be selected using the "`userspace`" governor are (in kHz): 2701000 (Turbo Boost), 2700000 (nominal frequency), 2400000, 2200000, 2000000, 1800000 1600000, 1400000, 1200000 (`/sys/devices/system/cpu/cpu*/cpufreq/scaling_available_frequencies`).

At hardware-level, the frequency is controlled by the processor itself and the P-states exposed to software are related to performance levels. Even if the scaling driver selects a single P-state the actual frequency the processor will run at is selected by the processor itself. In order to reduce energy costs, the processor may also shift one (or more) core and memory into lower power states (higher C-state) when idle, despite a P-state was selected by the OS. C-states available on the IVB-EP are C0 (active), C1 (halt), C3 (deep-sleep), C6 (deep power down). This phenomenon is observed in Appendix D.2.

2.3.2. Hardware performance counters

Most modern microprocessors provide hardware counters that monitor and report the count of hardware-related events concerning the CPU and its activity, including information like the elapsed clock ticks, instructions issued and retired, cache hits/misses, memory accesses and I/O read/write operations, which allow to obtain various derived metrics.
These hardware counters, called Performance Monitoring Counters (PMC), can be accessed using low-level calls to specific Configuration Space Registers (CSR), and can be used for the monitoring of the performance of the system, profiling of applications and their tuning. Various types of performance counter are implemented, covering different performance interests. Some counters can provide information regarding each single core of the CPU, some provide socket-wide information. “Socket” and “package” are sometimes used interchangeably, but relate to the same concept: everything available on the processor die. Other distinctions will be discussed in the following sections.

The IVB-EP processors support the following configuration register types:
- PCI Configuration Space Registers (CSR): chipset specific registers that are located at PCI defined address space.
- Machine Specific Registers (MSR), accessible by specific read and write instructions (*rdmsr*, *wrmsr*) accessible by OS ring 0 (the kernel mode with the highest privilege) and BIOS.
- Memory-mapped I/O (MMIO) registers: accessible by OS drivers.

The followings are the counters available on IVB-EP and directly or indirectly used in this work:
- per-core counters:
  - 3 fixed-purpose counters, each can measure only one specific event:

Counter name	Event name
FIXC0	INSTR_RETIRED_ANY
FIXC1	CPU_CLK_UNHALTED_CORE
FIXC2	CPU_CLK_UNHALTED_REF

  - 4 general-purpose counters, PMC<0-3>, each can be configured to report a specific event.
  - 1 thermal counter which reports the current temperature of the core.
- socket-wide counters:
  - Energy counters: provide measurements of the current energy consumption through the RAPL interface (see section 2.3.3).

Counter name	Event name
PWR0	PWR_PKG_ENERGY
PWR1	PWR_PP0_ENERGY
PWR2	PWR_PP1_ENERGY (not available on IVB-EP)
PWR3	PWR_DRAM_ENERGY

  - Home Agent counters (BBOX<0,1>C<0-3>): protocol side of memory interactions, memory reads/writes ordering (modular ring to IMC)
  - LLC-QPI fixed and general-purpose counters (SBOX<0,1,2>FIX, SBOX<0,1,2>C<0-3>): LLC snooping/forwarding and LLC-to-QPI related activities.
  - LLC counters (CBOX<0-15>C<0-3>): LLC coherency engine
  - UNCORE counters (UBOXFIX, UBOX<0,1>): measurements of the management box in the uncore (frequency of the uncore, physical read/write of distributed registers across physical processor using the Message Channel, interrupts handling)
  - Power control unit (PCU) fixed and general-purpose counters (WBOX<0,1>FIX, WBOX<0-3>): measurements of the power control unit (PCU) in the uncore (core/uncore power and thermal management, socket power states)

- Memory controller fixed and general-purpose counters (`MBOX<0-7>FIX`, `MBOX<0-7>C<0-3>`): DRAM related events (clock frequency, memory access, ...)
- Other Ring related counters: Ring to QPI (`RBOX<0,1,2>C<0-2>`), Ring to PCIe (`PBOX<0-3>`)
- IRP box counters `IBOX<0,1>C<0,1>`

Some of the aforementioned counters can be accessed through the MSR interface, for which Linux provide a specific driver and device, others through specific PCI or MMIO interfaces.

While per-core counters can be read from each core of the socket, socket-wide counters, like RAPL, can be accessed only from one of the cores (any one). `msr-statd`, the utility used to obtain the power consumption, was modified in order to use the PMU of the first core of each socket as source for socket-wide RAPL counters.

The general-purpose counters can be configured to read one of the hundreds of events supported by the processor[49]. Since there are 4 general-purpose counter available, only 4 events can be monitored at the same time (besides the fixed counters which are always available but not configurable).
In order to overcome this limitation, some utilities like `perf`, implement a multiplexing method that allows to switch the monitored events (PMU events only). With multiplexing, an event is not monitored continuously, but only for some repeated timed intervals, sharing the counter during the measuring period. At the end of the run, the aggregated event count is scaled for the complete period, thus providing an estimate of what the count could have been if it was measured for the whole run. Hence, scaled results are not completely reliable, as some blind spots may hide spikes, providing misleading results.

Counts reported by the fixed counters can be also obtained from equivalent configurable performance events on the general-purpose counters. Unfortunately, though, because of many factors, the values obtained differ, and this was one of the elements of confusion experienced during the test phase.

Each performance event is represented by an event number and a mask. In section 3 and appendices B and C, for all the events taken into exam, either the event name, the aliases or the hex flags (or all) may be reported.

Unfortunately, each generation of processors, and even different variants of the same model, can provide different counters and many different events. Sometimes new events are introduced, sometimes are removed, sometimes not implemented, and sometimes measure something different for each version. In the official documentation, sometimes the same metric is represented with multiple names and not always consistently. All this, and the differences introduced in the name spaces implemented in various monitoring and profiling software, make extremely difficult to pinpoint the exact meaning and usage of a specific event. Moreover, different utilities use different events claiming the same purpose. For sure, keeping such kind of software up-to-date for each new variant of processor and new innovation require a lot of efforts, and results can be only as accurate as the details available in the official documentation.
In order to provide a generic interface common for all the processors, Linux Perf/Kernel developers implemented some “aliases” for common events (unfortunately not always correct across different platforms), but kept open the possibility for a final user to specify explicitly an event to monitor. This feature was widely used in this work.

2.3.3. RAPL

Recent generations of Intel processors offer specific registers and counters which report, among thousands things, the power consumption of the CPU based on its main areas and functionality, like DRAM, CORE and UNCORE sub-systems. The UNCORE, called System Agent since Sandy Bridge, collects the functions of the microprocessor that are not in the CORE, but are essential for its performance. See figures 2.3.3:1 and 2.3.3:2.

Depending on the family/model, the sub-systems and the functions associated to each of them may vary. According to various (sometimes fuzzy) documentation[50][51][52][47][53], on Intel Ivy Bridge EP the sub-systems are divided as following:

- CORE: components of the processor involved in executing instructions, including ALU, FPU, L1, L2 and L3 cache;(*)
- UNCORE or System Agent: integrated memory controller (IMC), QuickPath interconnection (QPI), power control unit (PCU), ring interconnect, misc I/O (DMI, PCI-Express, ...).(*)

For desktop and mobile models, the Ivy Bridge processors may also include the Display Engine (included in the UNCORE/System Agent) and the integrated graphics processor (IGP).

The Figure 2.1:1 in section 2.1 (Hardware), shows the CPU internal topology (Core, L1i, L1d, L2, L3, DRAM) as reported by `lstopo` (`hwloc`).
Figure 2.3.3:1 shows the system topology.

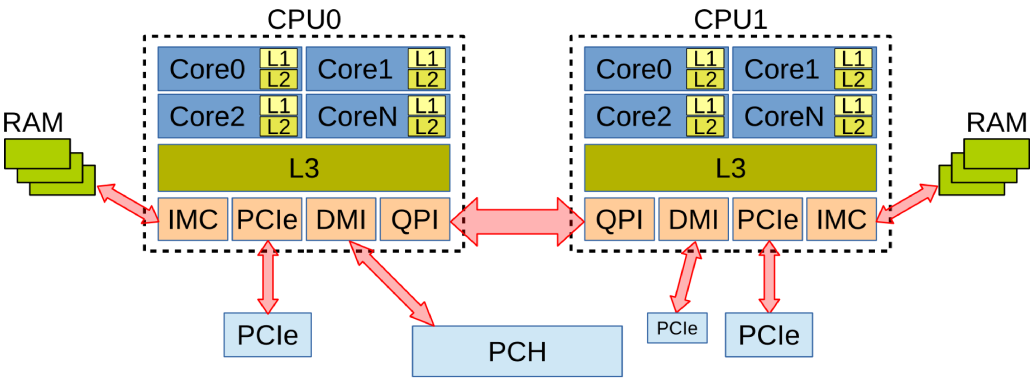


Figure 2.3.3:1: 2-way (2 sockets) IVB-EP System Topology
[adapted from various Intel references]

In RAPL[54], platforms are divided into domains for fine grained reports and control. A RAPL domain is a physically meaningful domain for power management. The specific RAPL domains available in a platform vary across product segments. Ivy Bridge platforms targeting server segment support the following RAPL domain hierarchy:

- Power Plane 0 (PP0): all cores and L1/L2/L3 caches on the package/die/socket (CORE)
- Package (PKG): processor die (PP0 + anything else on the package/die/socket (UNCORE))
- DRAM: directly-attached RAM

From the above, can be derived that UNCORE=PKG-PP0.
Each level of the RAPL hierarchy provides respective set of RAPL interface MSRs.

(*) for the acronyms, please refer to Appendix A.

RAPL “interfaces” consist of non-architectural MSRs. Each RAPL domain supports a set of capabilities:

- Power limit: MSR interfaces to specify power limit, time window, ...
- Energy Status: power metering interface providing energy consumption information
- Perf Status: interface providing information on the performance effects (regression) due to power limits (domain specific duration metric that measures the power limit effect in the respective domain).
- Power Info: Interface providing information on the range of parameters for a given domain, minimum power, maximum power etc.
- Policy: 4-bit priority information which is a hint to hardware for dividing budget between sub-domains in a parent domain.

Each of the above capabilities requires specific units in order to describe them. Power is expressed in Watts, Time is expressed in Seconds and Energy is expressed in Joules.

The “power info” RAPL interface reports the following power ranges for the platform tested:

PKG domain	DRAM domain
Thermal Design Power: 130 Watt	Thermal Design Power: 24.5 Watt
Minimum Power: 64 Watt	Minimum Power: 9 Watt
Maximum Power: 130 Watt	Maximum Power: 24.5 Watt

The “energy status” interface will be queried for power consumption information during the simulations described in the following sections (`msr-statd`).

The thermal design power (TDP), reported above, represents the maximum amount of power the cooling system in a computer is required to dissipate (if the cooling system is capable of dissipating that much heat, the chip will operate as intended). This is the power budget under which the system needs to operate. But this is not the same as the maximum power the processor can consume. It is possible for the processor to consume more than the TDP power for a short period of time without it being "thermally significant". Using basic physics, heat will take some time to propagate, so a short burst may not necessarily violate TDP. [13] [Figure D:8]

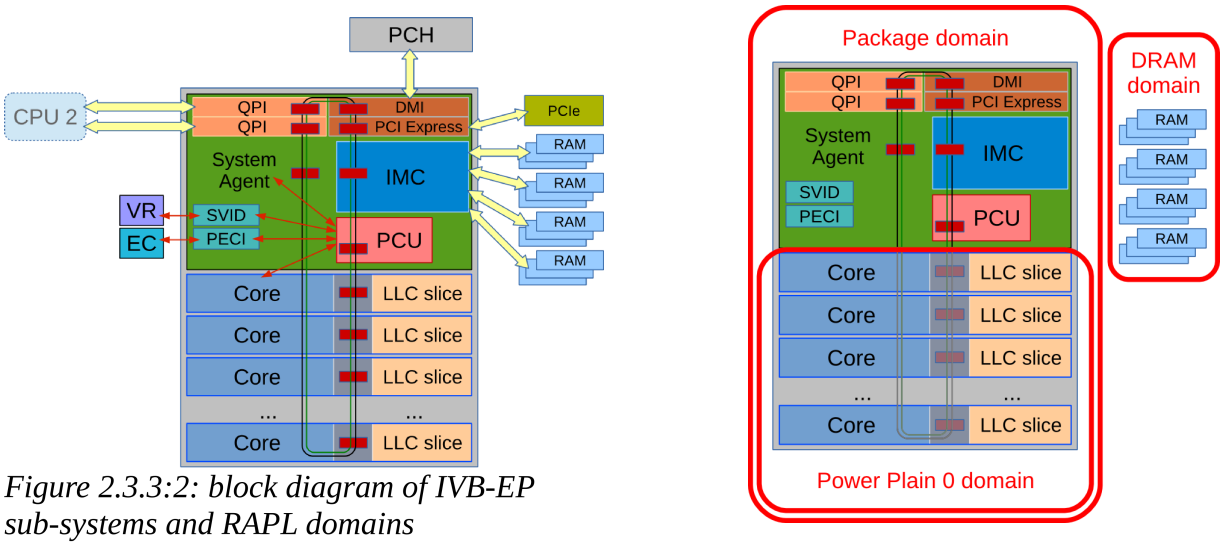
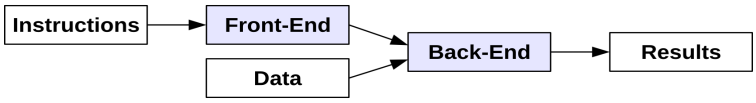


Figure 2.3.3:2: block diagram of IVB-EP sub-systems and RAPL domains [adapted from various Intel references]

2.3.4. Top-down characterization - TMAM

This section covers the Top-down Microarchitecture Analysis Method (TMAM)[55][56], a performance tuning technique which uses performance monitoring events specific to the Intel processors.

Modern microarchitectures implements what is called instruction-level parallelism. The CPU can execute multiple micro operations concurrently, in the same clock cycle, as long as they are busy in a different stage of the computation (traditionally fetch, decode, execute, memory access, write-back, but the stages are now many more).
In order to fill the pipeline and be efficient, each step must complete without stalling.

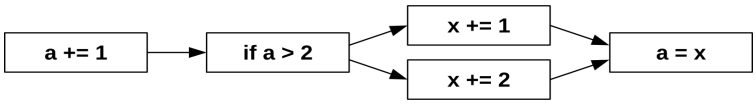


(image from [57])

If the front-end is unable to fill the pipeline by supplying enough instructions to satisfy the requests of the back-end, some clock cycles will be wasted without computing new micro-ops, therefore this situation is called front-end stall, and the execution is said to be *Front-End bound*.

If the back-end, instead, is unable to accept more micro-operations, the back-end stalls and the execution is said to be *Back-End bound*. The back-end can be delayed, for instance, by memory accesses. If the required data can be found and fetched from high-level caches, the delay can be minimal (few clock cycles), but if the data requires a fetch operation from the main memory, the back-end stalls, and this impacts the performance as the pipeline may be blocked.

Furthermore, in order to exploit the pipeline and avoid execution stalls, modern microprocessors implements two techniques called branch prediction and out-of-order execution, allowing to execute instructions speculatively ahead, just "guessing" what will be the outcome of the execution workflow.



(image from [57])

Bad speculation is said to happen when this guess is wrong, and in this case the cycles are wasted, as the result of those operation won't be used and is therefore discarded. When this happens, the pipeline must be flushed, wasting even more clock cycles.

When an instructions completes this process without being bottlenecked is said to be *retiring*.

Hierarchically, the execution opportunities described above can be logically divided as illustrated in Figure 2.3.4:1. In greater details, Figure 2.3.4:2 exposes the complete characterization of microarchitectural issues.

By using the performance counters related to the execution flow, it is possible to estimate the amount of execution slots spent in each category, hence identify which is more likely the area of microarchitecture to investigate for bottlenecks.

Very good and comprehensive explanation of the methodology can be found at [58] [57] [59] [56].

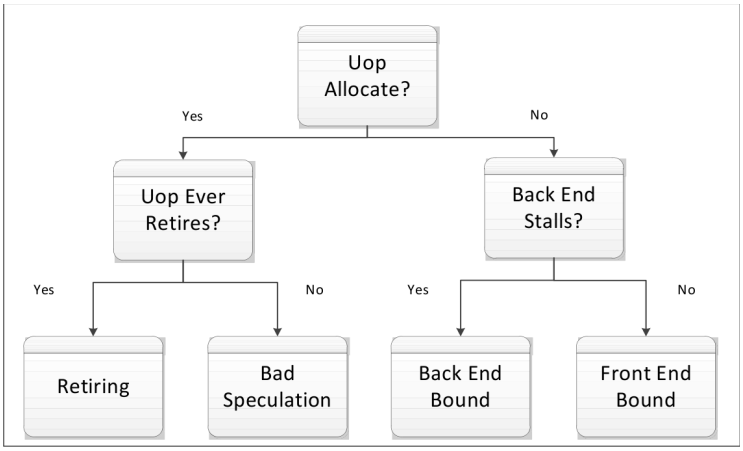


Figure 2.3.4:1: Pipeline Slot Classification Flow Chart [55]

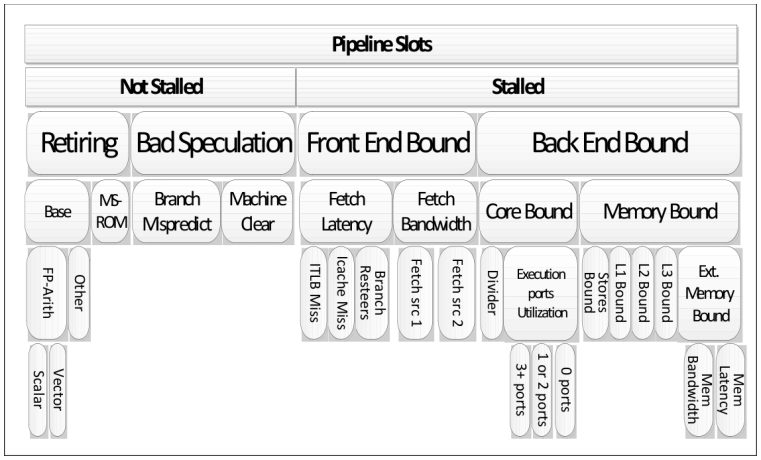


Figure 2.3.4:2: General TMAM Hierarchy for Out-of-Order Microarchitectures [55]

To summarize, a program can be bound to I/O (memory, storage, network), or can be bound to the CPU or other components, such as the GPU. Being bottlenecked by the front-end means that the pipeline can't be filled, being back-end bound means that for some reasons the pipeline is stalling, usually waiting for data from the memory. Bad speculation is basically due to the branch misprediction. The retired instructions are those that completed successfully.
In this work, this analysis will be used to identify memory-bound and CPU-bound applications.

According to [59], depending on the kind of workload, different range of pipeline slots can be expected for each category. The following table reports the typical distribution for a well-tuned HPC application:

Retiring	30-70%
Back-End Bound	20-40%
Fronte-End Bound	5-10%
Bad Speculation	1-5%

In Appendix B. details on how to compute the pipeline slots are provided.

2.3.5. Tools

A large variety of tools, utilities, frameworks and libraries have been developed in the last few years which permit to read and act on the CPU registers or allow to enable user-space applications to offer power and performance monitoring capabilities.
Citing few relevant ones among those investigated in a preliminary study for this project:

- `perf`[41][60] (Linux profiling with performance counters) is a performance analyzing tool in Linux which provides user-space controlling utility, an event-driven interface in kernel space, and it is capable of statistical profiling of the entire system (both kernel and userland code), supporting hardware/software performance counters, tracepoints, and dynamic probes.
- `LIKWID`[34][61] (lightweight performance tools for x86 multicore environments) can probe thread/cache topology of a shared-memory node, enforce thread-core affinity on a program, measure performance counter metrics, access RAPL counters and query Turbo mode steps on Intel processor (`likwid-powermeter`, `likwid-perfctr`).
- `libpfm4` (`perfmon2`)[62] is a helper library to help encode Performance Events to use with operating system kernels performance monitoring interfaces (`perf`).
- `pmu-tools`[63] is a collection of tools for profile collection and performance analysis on Intel CPUs on top of Linux `perf`.
- `PAPI`[64] (Performance Application Programming Interface) aims to provide a generic abstraction layer in order to interface user-space applications to the performance counter hardware, relating software performance and processor events, and collecting components that expose performance measurement opportunities across the hardware and software stack.

Other tools include: msr-tools[65], perfmon[66], powertop[67], turbostat[68], cpupowerutils[69] and cpuspeed[70].

Bypassing this tools, sometimes necessary, requires specific code to be written using low-level system calls and passing utterly cryptic flags and parameters only reported (although not always explained in details) on microprocessor's development manuals.

In order to enrich the poor environment of `perf` (see command “`perf list`”), which implements 49 shortcuts over thousands flags, some efforts was taken during this project to convert automatically `LIKWID`, `libpfm` and `pmu-tools` specific flags (over one thousand) into a format compatible as input for `perf`. This operation led to the discovery of repeated event flags accessed using different event names, or the same event name mapped to different event flags in different programs.

It must be noted that Linux Perf implementation in recent kernels exposes RAPL to user-space (through `perf` and `sysfs`), as well as part of the top-down analysis (`perf`). Unfortunately, due to a large number of dependencies, upgrading the kernel in a production cluster is not always feasible, like it wasn't in this case.

3. Results

In this section a selected collection of the results produced during this work will be exposed and commented in separate subsections, each devoted to specific tests and relative analysis. Several plots accompany the most relevant results obtained.

This work deals with an almost unmanageable number of degrees of freedom because of all the performance events, metrics, and measures gathered. Not all the data collected could be presented and a large amount had to be ignored or discarded. Additional plots deemed not-essentials, even though cited, are available in Appendix E., while Appendix D.3. gives a brief and rough idea of the data collected and the kind of information that can be extracted.

Energy measurements are performed and discussed in this section and it is therefore important to specify which kind of the metrics and quantities can be observed and measured. This is the goal of section 3.1.

The following table provides a concise overview of the test performed and exposed in the following sections, to offer a guidance to the reader through the chapter.
The table does not include additional tests performed only on some specific benchmark, analysis of which will be presented and discussed in appropriate sections.

category	test	HPL				HPCG	QE	LAMMPS
		Netlib	ATLAS	OpenBLAS	MKL			
energy efficiency	basic analysis & tuning (*)	x	x	x	x	x	x	x
	frequency scaling	(**)	x	x	x	x	x	x
	problem size scaling	(**)	x	x	x	x		x
performance counters (*)	top-down analysis	x	x	x	x	x	x	x
	cache-miss, branch mispr.	x	x	x	x	x	x	x
	FLOPS from counters				x	x	x	x
	exp. cache-miss (***)	x	x	x	x			

(*) with `ondemand` governor (automatic frequency scaling w/ Turbo Boost)
(**) only for memory size 1/4 and 1/8
(***) experimental tests using various combinations of cache-related performance events (Appendix F.)

Section 3.2 reports the extensive analysis performed on the HPL benchmark. As previously discussed, HPL was chosen due to the relative simplicity and for the detailed information it provides. However, by just changing external BLAS libraries, a wealth of measurements are possible, thus enriching the power measurement analysis and its comprehension.

Section 3.3 reports the analysis on HPCG. HPCG is a fairly new tool, not widely used yet, but it aims to become an alternative benchmark to HPL for performance assessment and ranking. Unlike

HPL, which may need hours to complete (Netlib test), HPCG can be told to run for a specific amount of time. Limiting energy waste and allowing for a larger number of tests, it still gives reliable results even for short runs of one or few minutes.

Sections 3.4 and 3.5 review the same methods applied to two representative real-world applications used by the scientific community, Quantum ESPRESSO and LAMMPS. The input file for QE is part of a research currently undertaken by CNR-IOM scientific research group. Concerning LAMMPS, one of the benchmarks provided by the software package was chosen, the Lennard-Jones liquid benchmark[71], even though it was scaled up from the original in order to exploit the available memory and CPU resources.

Some selected results of above sections are then collected in section 3.6.3, which concludes this chapter, where a detailed comparison among different benchmarks sessions is conducted.

It should be remarked that all the benchmarks reported in this section use a single computing node. Due to the lack of additional sensors and devices able to indicate reliable power consumption measurements of surrounding devices, this study focused on RAPL readings obtained from the two processors available on each node. Multinode testing (Appendix D.1.2.) shown that MPI communication indeed impacts the walltime. Properly estimating how this communication may affect the overall energy consumption, though, would require additional metrics concerning network-related activities and power consumption measurements of adapter cards and network devices.

3.1. Energy measurement

Although many metrics exist to describe various aspect of the energy efficiency, this works focused on the followings:

Overall energy (J)
Average power (W)
FLOPS per Watt
FLOPS per Joule

The relationship between energy and power is of course given by:

E_{tot} = P_{avg} * walltime

P_{avg} = E_{tot} / walltime

Here, walltime is intended as the total running time of the application, as reported by the application itself, when implemented, or by the wrapper program (`msr-statd`, `gpu-statd`, `perf`, `bash time`, `/usr/bin/time`).

The RAPL interface provides incremental readings in Joule, for each RAPL domain, for each socket. `msr-statd` collects these readings in a timely manner and save this raw data into a file for later analysis. By post-processing the produced log file, it is possible to obtain the total energy, the runtime period, and from these derive the average power (for each socket). Additional metrics, less relevant in this case, include core temperature, frequency and load. A brief overview of the ignored metrics are reported in Appendix D.
In the results exposed in this work, the aggregate energy and power for both sockets is considered, even though split by RAPL domains.
Unless otherwise stated, all the benchmarks were run considering a single node, using both the available sockets and all 24 cores. For the performance counters analysis and top-down characterization, the `ondemand` frequency scaling governor was active and Turbo Boost enabled. The energy efficiency study was conducted with the `userspace` frequency scaling governor and fixed imposed frequency. Hyper-threading (SMT) was always disabled.

It has to be remarked that the software benchmarked was considered a black-box, no investigation has been done on the algorithms, nor on the time-based evolution of power-consumption (although possible as reported in Appendix D.3.). The analysis performed in this work was meant to be a proof of concept for an unattended procedure integrated into an energy-aware scheduler.

Finally, it has to be noted that:

1. HPL reports the performance obtained (GFLOPS) and the time-to-solution (seconds). After the actual computation, though, it performs an additional check in order to validate the results (not counted in the values reported). This operation increases the walltime as detected by external wrappers, thus it affects the analysis. The overhead of this operation is minimal in respect of the walltime, but it is large enough to alter the results when used in conjunction with the reported FLOPS. In order to overcome this issue, due to the negligible

impact on the energy consumption represented by the final check over the whole run time, it was deemed reasonable to consider the P_{avg} as a reliable estimation and use the time-to-solution (as reported by HPL) in order to obtain the overall energy consumed:

P_{avg} = E_{tot} / walltime

energy-to-solution = P_{avg} * time-to-solution

2. QE and LAMMPS don't provide the achieved performance in FLOPS. Experimenting with performance counters, an attempt to estimate this metric was made by using the performance events and formulas shown in Appendix C.

In order to identify the effects of fluctuations due to load and OS environment "noise", various benchmark were run multiple times. Effects of fluctuation was observed and deemed negligible for HPL and HPCG tests. QE and LAMMPS shown larger fluctuations, hence the average of 10 runs is reported. The error bar is estimated within a few percent for HPL and HPCG, 7% (max) on total walltime of Quantum ESPRESSO, 20% (max) on total walltime of LAMMPS.

3.2. HPL

This section examines performance, power consumption and energy efficiency of HPL compiled against four different implementations of BLAS as provided by the following libraries:

- Netlib, reference library, accurate but not optimized;
- ATLAS, auto-tuned library, optimized;
- OpenBLAS, highly optimized open-source version;
- Intel MKL, proprietary and closed-source version, highly optimized for Intel processors.

Should be noted that this analysis does not aim to provide a full-fledged comparison of the entire libraries or all BLAS routines, only the BLAS functions invoked by HPL are relevant in this case, with a net predominance of the GEMM family.

The first step in this phase was a tuning session in order to identify which are the best parameters in terms of problem sizes (Ns), block sizes (NBs) and grid distributions (Ps, Qs), that provide the best performance.

The best performing set of parameters turned out to be the following for all the libraries:

Ns = 81920
NBs = 176
Ps = 6
Qs = 4

which corresponds roughly to 54 GB of RAM utilization (problem size = $N^2 * 8$).
These settings allowed to obtain 480 GFLOPS (using HPL+MKL) over a theoretical peak performance of 518.4 GFLOPS (92.6%).

The parameters identified in this tuning runs are then used for all the runs later reported in this section unless otherwise specified.

The same methodology was adopted for the tuning of all the benchmarks tested, where a preliminary session was devoted to obtain significant data sets, possibly with acceptable runtimes (<30 minutes), and later adopted for the size and frequency scaling analysis.

3.2.1. Comparing BLAS implementations

The graph in Figure 3.2.1:1 reports the performance, walltime, average power, total energy and energy efficiency, expressed as GLOFPS/W and MFLOPS/J, for the four implementations. This plot reports a stacked view of the contribution given by each CPU sub-systems. As a reminder, the CPU package/socket (PKG RAPL domain) is made by CORE (PP0 RAPL domain) and UNCORE (PKG – PP0). DRAM is the physical memory.

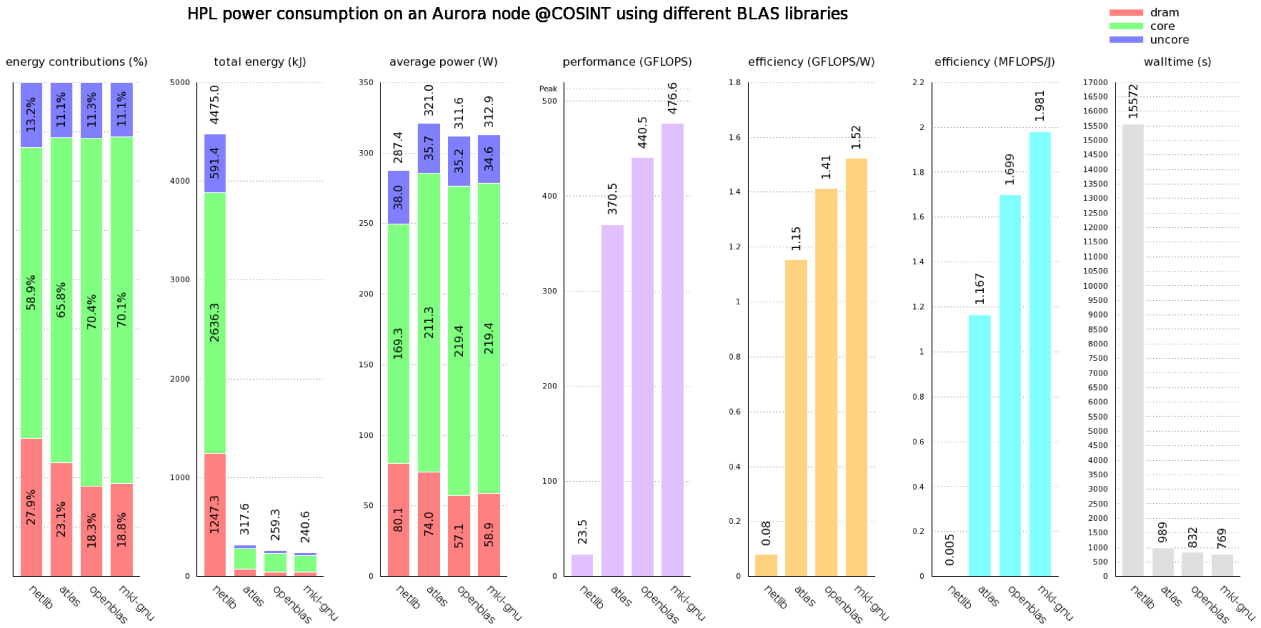


*Figure 3.2.1:1: BLAS comparison (*ondemand *governor, Turbo Boost, optimal size)*

Figure 3.2.1:1 clearly indicates that the delivered performance is very different. The walltime of course influences the overall energy consumption, but as well the CPU utilization affects the efficiency: the denser the computation is in the CPU (less stalls due to memory access), the larger the efficiency (MFLOPS/W).

Walltime and power consumption of Netlib are 2 order of magnitude greater than the optimized versions, which makes it the least performing and the least energy efficient library.
MKL and OpenBLAS, though, are quite close to each other. The average power is the same, but the slightly different performance, and therefore larger walltime of OpenBLAS, influences the total amount of energy consumed, and thus the efficiency.

The results have been further split by CPU sub-systems (DRAM, PKG, CORE, UNCORE) and the plots reported in Appendix E.1.
Figure 3.2.1:2 reports a detailed view of the energy distribution by sub-systems.

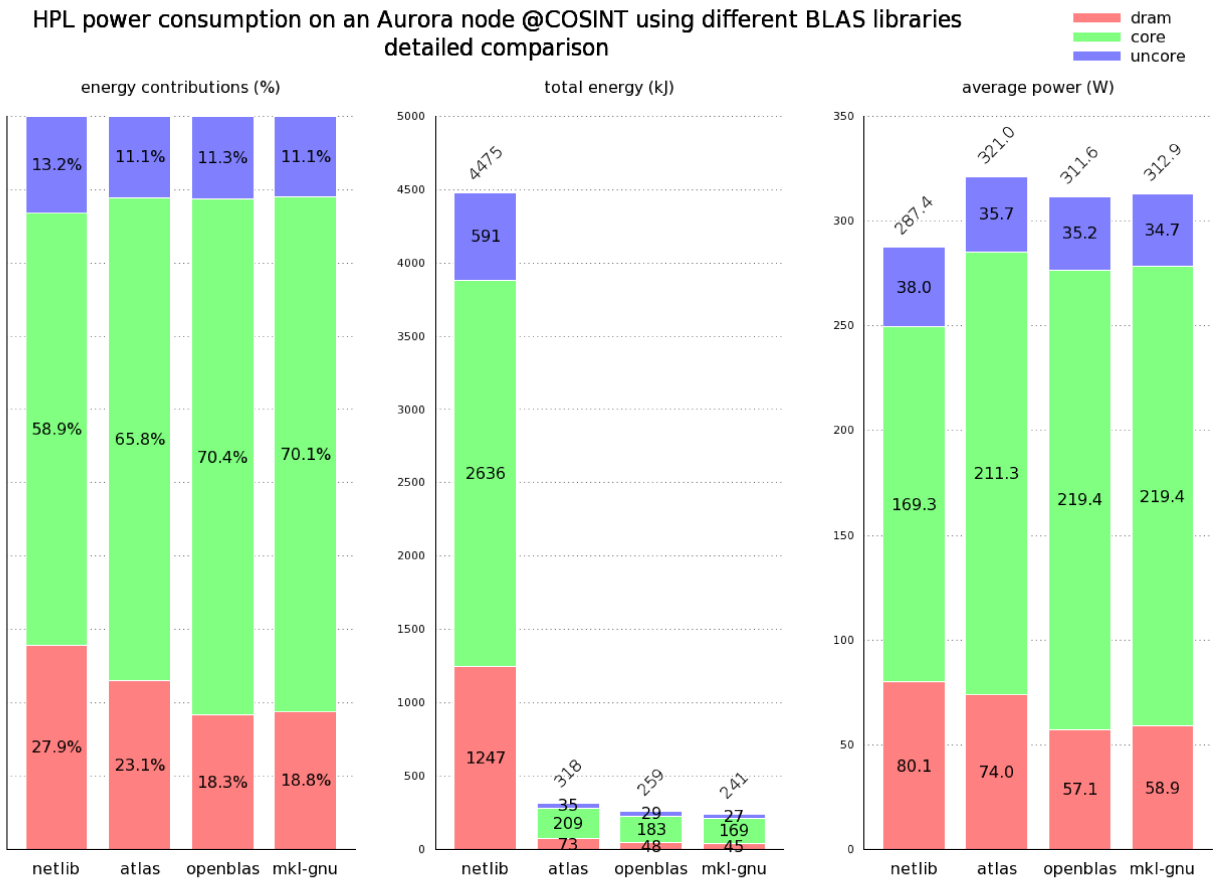


Figure 3.2.1:2: BLAS comparison, detailed view

It can be observed that using different implementations may lead to a different distribution of the power consumption. Netlib, for instance, appears to be more memory-bound than the others, hence the calculation consumes more power on the DRAM domain. Because of this pattern, the average overall-power is lower, as the CPU is probably idling longer while waiting for data. OpenBLAS and MKL, on the other end, are very effective at core level, denoting a more efficient utilization of the memory hierarchy.

More details on this phenomenon can be obtained by performing a deeper analysis of the issue using the performance counters and a top-down characterization, which allows to identify the stalls and bottlenecks in the pipeline. This will be performed in the following subsections.

3.2.2. Top-down characterization

Here, the results of the top-down characterization are presented. The purpose of this analysis is to identify the bottlenecks in the instruction pipeline.

As explained in section 2.3.4, the optimal BE-bound value for an HPC application is expected to be within 20% and 40%.

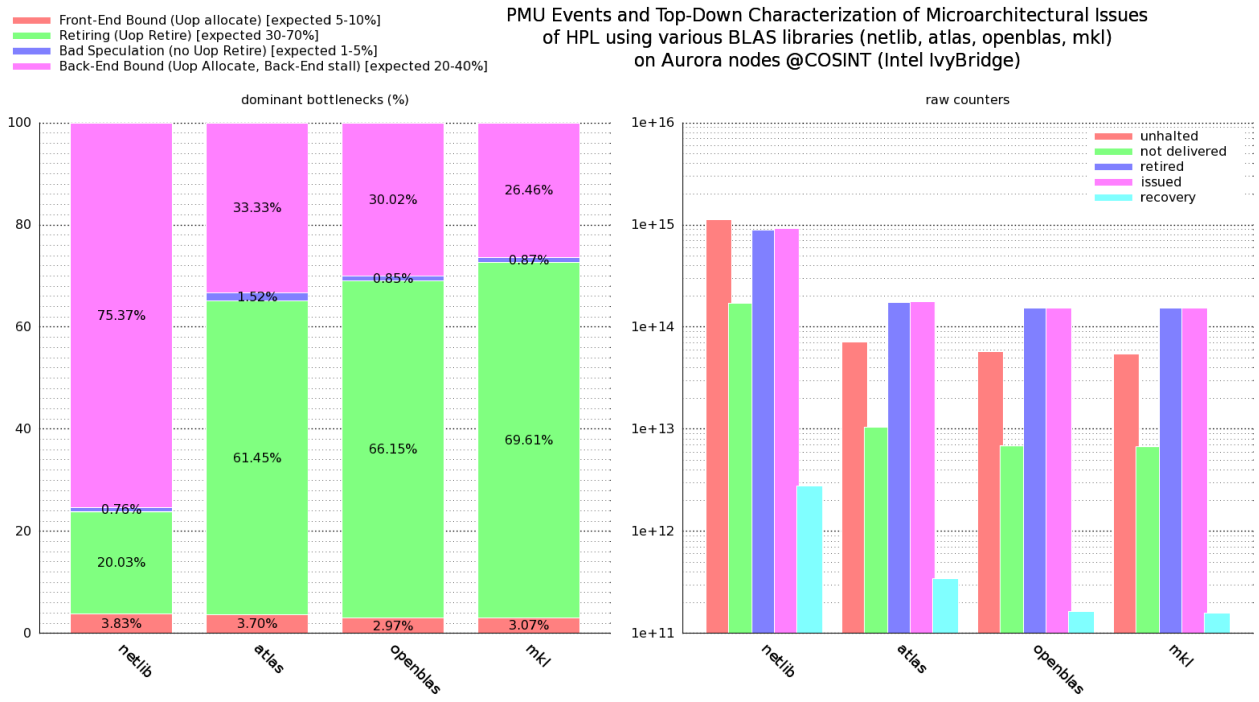


Figure 3.2.2:1: BLAS comparison: TMAM

Looking at Figure 3.2.2:1, the following observations can be made:

- Netlib reaches the 75% on the BE-bound value. From this analysis is clear how Netlib is back-end bound, which implies a problem related to the memory access that is bottlenecking the application. Probably, a sub-optimal memory access pattern makes it unable to take advantage of the new architecture's caches, hence preventing it to exploit the pipeline efficiently.

- the other implementations, are comfortably inside the optimal limits for a typical HPC application.

3.2.3. Performance counters

The following analysis compares the readings gathered from the performance counters using `perf`. The metrics considered relevant for this first analysis are:

- *cpu-cycles*
- *instructions*
- *cache-references*
- *cache-misses*
- *branch-instructions*
- *branch-misses*

The last 4 parameters of the list permit to obtain 2 derived metrics, the *cache-miss ratio* and the *branch-misprediction ratio*. The first is an index of how well the application exploits the cache hierarchy. The second concerns the efficiency in guessing what will be the next instruction to execute in case of conditional jumps (branch). In order to exploit the superscalar characteristics of the pipeline, the CPU preemptively fill the pipeline and executes instructions from multiple branches (or just guesses one), long before the branch true execution path is known. If the next instruction is guessed wrong (the conditional statement eventually pointed to another branch), the result will be discarded and some CPU cycles wasted.

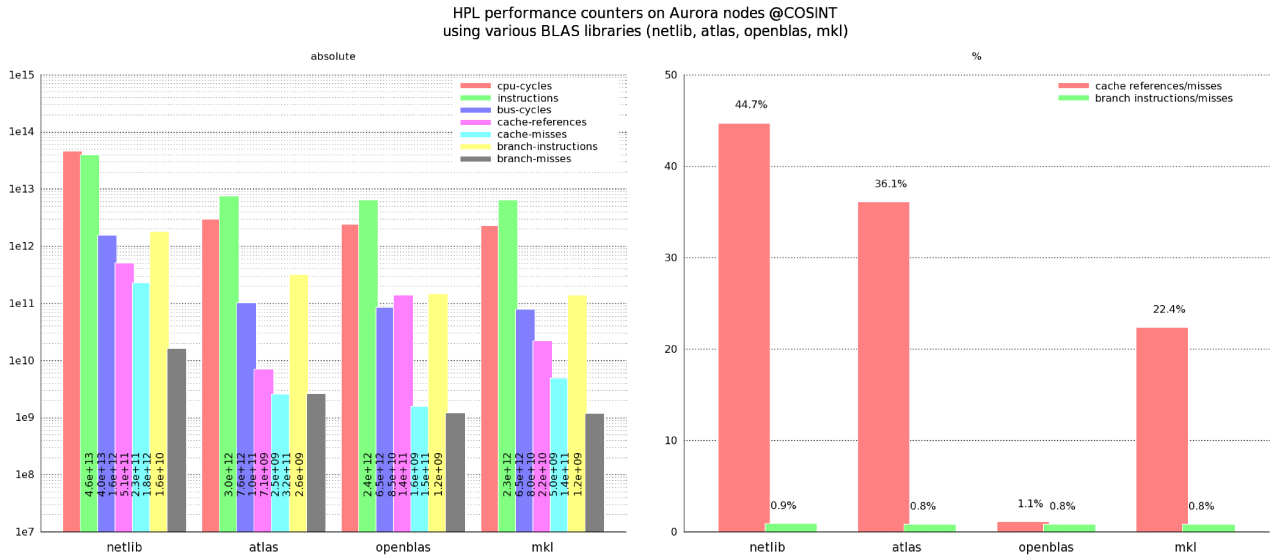


Figure 3.2.3:1: BLAS comparison, performance counters

Figure 3.2.3:1 compares both raw counters (left panel) and derived metrics (right panel) for each BLAS version.

As can be seen, the branch misprediction ratio (the green bar in the rightmost graph) maintains constant across all the implementations. The behavior is probably related to the main algorithm, not to the actual portion of code devoted to the computation. Further investigation is in any case required, but time constraints made the study focus on other more significant metrics.

The red bar in the right panel of 3.2.3:1, presents the cache-miss ration. At first sight, OpenBLAS cache-miss ratio is astounding, showing an incredibly effective utilization of cache accesses. The test was repeated twice in order to verify such results, confirming the behavior, but it looked suspicious nonetheless.

By reading thoroughly various sources of documentation[2], it turned out that the parameters named "*cache-misses*" and "*cache-references*" used by `perf`, are translated to the following performance events in Intel processors:

```
                      perf                                |    event
alias (u-space)  define (k-space)                hexcode|code mask name
cache-misses     PERF_COUNT_HW_CACHE_MISSES      0x412e |2Eh  41h  L3_LAT_CACHE.MISS
cache-references PERF_COUNT_HW_CACHE_REFERENCES 0x4f2e |2Eh  4Fh  L3_LAT_CACHE.REFERENCE
```

Accordingly to Intel's manual[74] (tables 19-17 and 19-19):

2EH 41H L3_LAT_CACHE.MISS
This event counts each cache miss condition for references to the last level cache.
The event count may include speculative traffic but excludes cache line fills due to L2 hardware-prefetches.

2EH 4FH L3_LAT_CACHE.REFERENCE
This event counts requests originating from the core that reference a cache line in the last level cache.
The event count includes speculative traffic but excludes cache line fills due to a L2 hardware-prefetch.

And for both:
Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

Thus, it seems that using LLC references for the overall cache-miss ratio is not accurate as it represents only the memory accesses that hit LLC while does not include cache hits in L1 and L2.

Looking at raw counters (left panel of Figure 3.2.3:1), it appears like L3 references in OpenBLAS increase by an order of magnitude over MKL and ATLAS, while L3 misses decrease by almost an order of magnitude, thus leading to the 1% of cache-miss ratio versus the 22% delivered by MKL.
Consequences of this may be one of the following:

1. L3_LAT_CACHE events cannot be thoroughly trusted;
2. OpenBLAS is optimized to exploit L3 locality, storing to and fetching from the LLC.

Various tests, briefly reported as a few plots in Appendix F., were later performed in order to obtain alternative performance events from which L1/L2/L3 cache-miss ratios can be derived.
Unfortunately, various combinations adopted by various tools (`LIKWID`, `PAPI`, `libpfm`) or found on the web and tested here, led to completely different results for what should represent the same metric.
Many factors should be taken into account, though:

- some hardware counters offer speculative counting of performance events (instead of real);
- `perf` multiplexing and scaling cannot report exact readings when too many events are concurrently monitored (because only 4 configurable hardware counters are available);
- the documentation sometimes reports about counters that are reliable for some specific architecture, but does not explicitly mention whether they are reliable too for other architectures (and sometimes it is not consistent) -- inheritance should (or should not) be blindly assumed;
- some applications (and web forums) take for granted that what was working on a specific processor can be extended to the same family of processors;
- workarounds are sometimes needed to deal with *errata*.[75][76][77][78][79][80][81]

It is therefore really hard to tell which combination can be trusted, and on which platform, since there's no other mean to verify the results.

2 Intel's processors developers manuals[50][51][52] and the source code of `perf` (both user-space[72] and kernel-space[73]), `PAPI`, `LIKWID`, `libpfm4` and `pmu-tools`.

3.2.4. Frequency scaling

This section compares the behavior of the application when the CPU operating frequency is scaled.

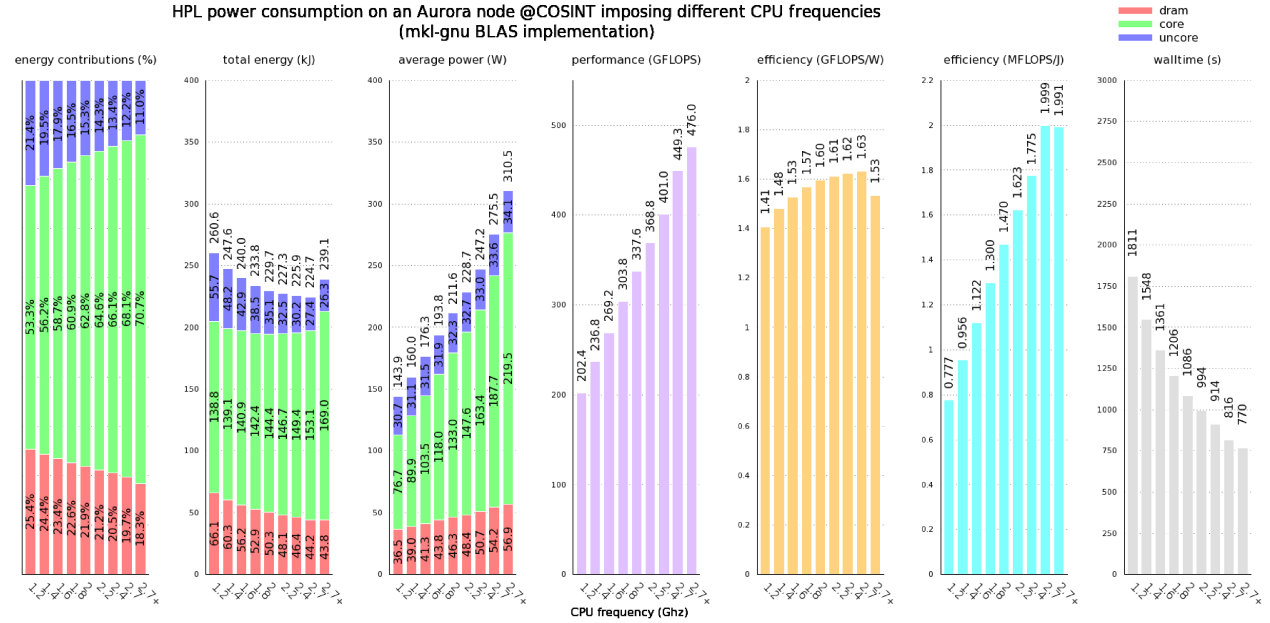


Figure 3.2.4:1: BLAS comparison, MKL BLAS frequency scaling

Figure 3.2.4:1. shows a comparison of the performance of HPL+MKL by imposing different frequencies.
In this case, the processor supports frequencies from 1.2 GHz to 2.7 GHz. The additional label 2.7+ represents the Turbo Boost, which can span from 2.8 to 3.5 GHz depending on the core temperature and the number of cores active:

3500 MHz (1 core)	3200 MHz (4 cores)
3400 MHz (2 cores)	3100 MHz (5 cores)
3300 MHz (3 cores)	3000 MHz (6 or more cores)

For HPL, being a CPU-bound application, when the frequency is increased the performance increases too, and as can be seen, the increase is almost linear. This means that the application benefits from running at higher frequencies. The power consumption, though, increases too.
Two interesting observations can be done looking at power efficiency (panel 5) and energy efficiency (panel 6). Power efficiency reaches the maximum at nominal speed of the processor (2.7 GHz), while when Turbo Boost is enabled there is a small drop. However this drop in power efficiency is not present when energy efficiency metric is considered: in this case, Turbo Boost allows to run faster without impacting the energy efficiency because the performance rises at the same rate of the overall energy consumption.

The results for the other BLAS implementations are almost identical. Frequency scaling plots for Netlib, ATLAS and OpenBLAS are available in Appendix E.3.

3.2.5. Problem size scaling

In this test the problem was scaled down to 1/2, 1/4 and 1/8 of the memory used by the best performing combination (~54 GB, ~83% of the total RAM).

The goal was to identify whether varying the amount of memory used would change the distribution of energy consumption during the run and therefore the energy-efficiency.

HPL is expected to deliver worse performance when the size is decreased. With the size, though, also the time-to-completion decreases. This, combined with the lower utilization of the CPU, may lead to a lower average power, and lower overall energy consumption. If the power consumption decreases faster than the performance, the energy-efficiency may be higher for less performing runs.

As can be seen from Appendix E.2., HPL behaves as expected. The distribution doesn't appear to change, in fact CORE, UNCORE and DRAM contributions appear to be the same for all the runs. Nonetheless, the average power is slightly different, as it decreases for the CORE when decreasing the size. The large difference in the time-to-completion impacts the total energy, and this the energy-efficiency.
ATLAS, MKL and OpenBLAS behave more or less the same, and exhibit a better average power efficiency (GFLOPS/W) with the larger size. Netlib, shows instead the opposite. Once again, this behavior is probably related to the memory access pattern, as it delivers the best performance with the smallest size.

Nevertheless, once again the time-to-solution affects the final outcome. Along with the decrease in size, there is a very small decrease in the average power consumption, but the walltime decreases faster than the performance, therefore the overall energy efficiency (expressed as MFLOPS/J) is higher with the smaller size. For Netlib, this increment is even larger than for the other libraries.

The same test was performed also by scaling the frequency, confirming the behavior observed in the previous tests and already described. The plots are available in Appendix E.3.

3.2.6. HPL + GPU

An exploratory testing was performed by using the hybrid CPU/GPU version of HPL as supplied by NVidia. The goal was to observe the effects of the CPU frequency scaling on the overall power consumption, including the 2 GPUs. GPU frequency scaling was not approached.

Power consumption information of the GPUs was obtained by means of the NVidia Management Library (NVML)[35]. A python script (`gpu-statd`), part of the Eurora monitoring framework, was adapted and used to retrieve the data.

Various tests pointed out the best performing inputs, obtaining 2.3 TFLOPS out of the combined (CPU+GPU) 2.8 TFLOPS peak performance (~82%):

Ns = 85000
NBs = 896
Ps = 2
Qs = 1

Should be noted that many other combinations, even though performing better, consistently failed the final validation check, and were therefore discarded.

This version of HPL allows to define the portion of computation to perform in the GPU. A wrapper script was written for the correct setup of the environment.
In order to avoid migrations and optimize the hybrid parallelism using MPI/OMP and the GPU, 2 MPI processes was used, each bound to a specific CPU (using `numactl` from inside the wrapper script instead of `mpirun --bind-to`). Each MPI process spawned 12 OMP threads (1 for each core of the CPU). Only one GPU was made visible to each MPI process. 90% of the computation was requested to be run on the GPU.

Figure 3.2.6:1 reports the aggregated power consumption of the 2 GPU only (no CPU).
Figure 3.2.6:2 reports the overall power consumption, including the CPUs.
The idle power measured on each GPUs was 41 W, the total power reached during the setup phase and the following tests was 192 W.

In Figure 3.2.6:1, these tests evidenced the fact that the frequency at which the CPU operates can affect as well the performance obtained on the GPU. In this case, the CPU cannot feed adequately the GPU, which stalls, hence the larger walltime when running at lower frequencies. The almost uniform average power consumption makes again the total energy dependent on the time-to-solution. Nonetheless, it can be noted that at 2.2 GHz and 2.4 GHz the efficiency is close or even larger that what can be obtained at 2.7 GHz. This is even more evident in the aggregated report in Figure 3.2.6:2.

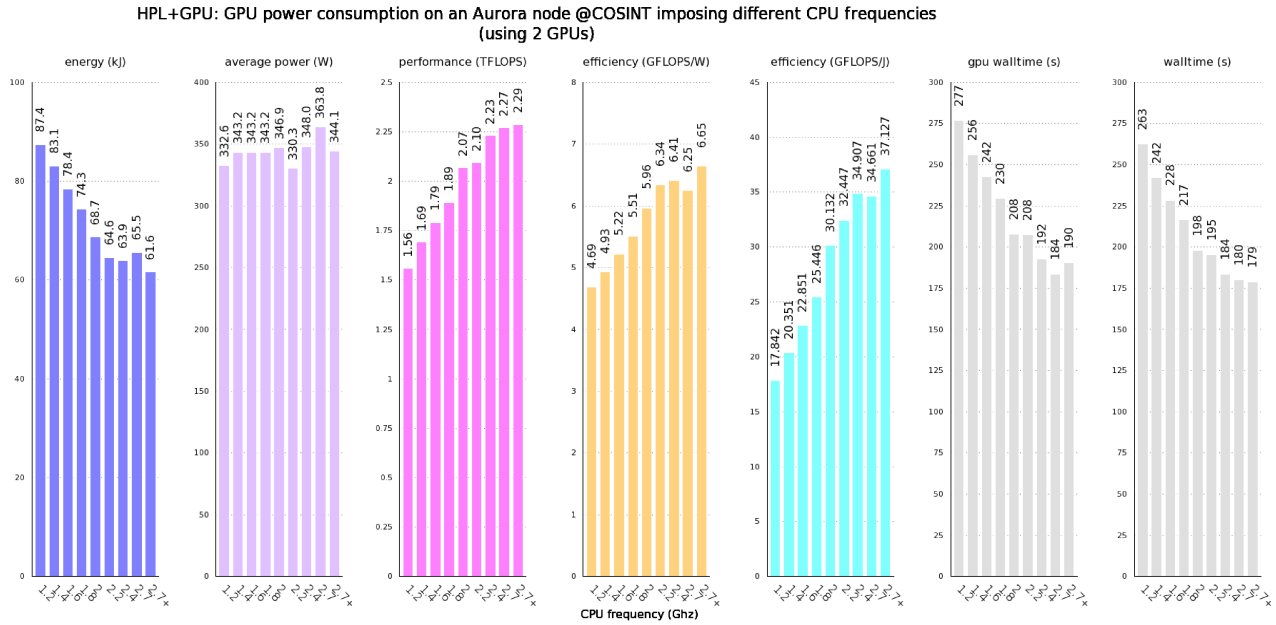


Figure 3.2.6:1: GPUs aggregated power consumption

As can be noted, from 3.2.6:2, more than 60% of the overall power is consumed by the GPUs. The effects of the frequency scaling on the power consumption of the CPUs is added to what observed before in 3.2.6:1. The runs at 2.2 GHz and 2.4 GHz appear to be the more convenient from the energy-efficiency point of view, as the energy-to-solution is lower than what can be obtained at any other frequency.
In terms of pure performance, by running at 2.4 GHz less than 3% is lost, but in terms of energy, the 6% is spared.

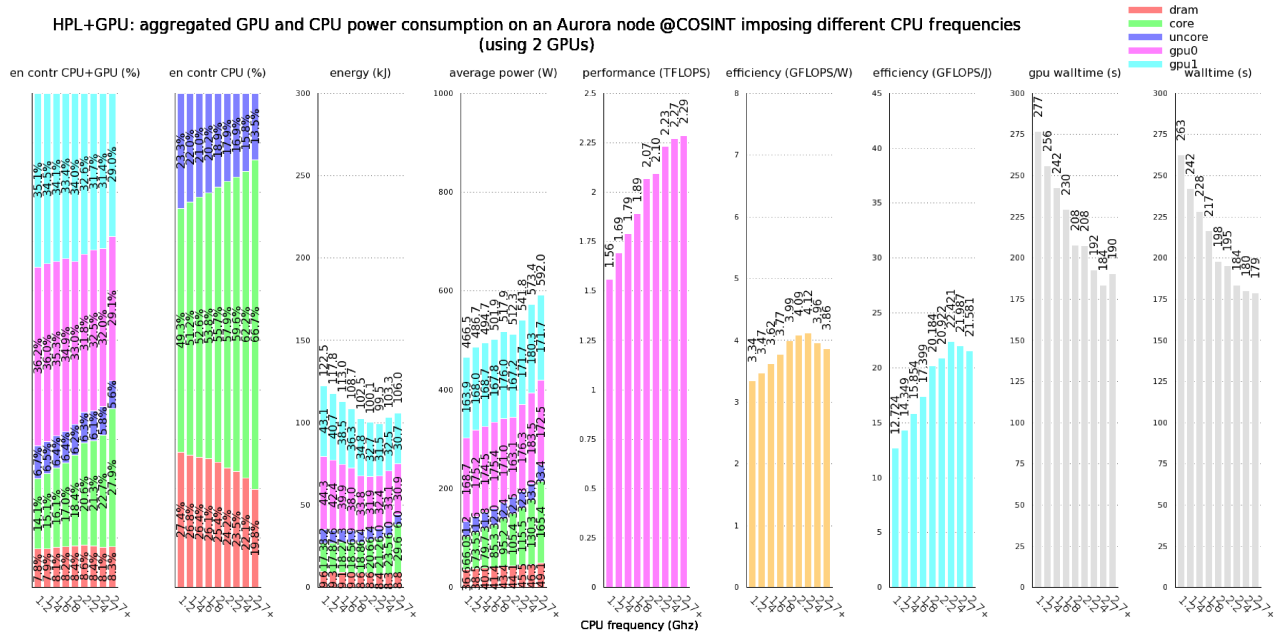


Figure 3.2.6:2: overall power consumption and energy contributions

Should be noted that the peak performance obtained, 2.292 TFLOPS with a power consumption of 592 W, would be ranked at the 4th position of the Green500 (Nov 2015) with 3.87 GFLOPS/W. Unfortunately, the power measure took into account only the CPUs and the GPUs of a single node, but even considering twice as much power (unlikely), it would be still ranked 82nd.
The cluster ranked at the 500th position of the Green500, QUARTETTO[8], in order to deliver the same performance would have consumed 43 kW, a 73 times larger power consumption.

Oddly enough, the most energy-efficient combination was not the best performing run. By running at a lower CPU frequency (2.4 GHz) and obtaining a slightly worse performance (2.23 TFLOPS), the average power consumption was smaller (541.8 W), 50 W less than the best performing run (10%).

3.2.7. Summary of HPL investigation

The results of the HPL investigation can be summarized as follow:

- MKL and OpenBLAS delivered the best performance, even though the pattern found in the cache-miss ratio looks quite different;
- using an unoptimized library (Netlib in this case) may lead to catastrophic results, both in terms of overall performance and energy-efficiency;
- not surprisingly, for a CPU-bound application the performance achieved is linear with the frequency scaling;
- GPU performance are unexpectedly driven by CPU operating frequency;
- problem size scaling behavior of Netlib is in counter tendency in respect to the other libraries;
- 4th position in the Green500!!! Almost...

3.3. HPCG

In this section results from HPCG are presented.
During the setup phase various test were conducted to check the best performing combination and validate the results. HPCG, unlike HPL, allows to set the desired walltime, thus simplifying the calibration. The runtimes tested were 30, 60, 120, 180, 240 and 300 seconds. The problem size was scaled up from 100 MB to 215 GB, using X=8, 16, 32, 64, 104 (total size = X^3 * 8 * NP).
As for HPL, the frequency was scaled for each step allowed by the CPU.

As previously observed for HPL+Netlib, a memory-bound application doesn't benefit of the peak performance of the processor. Hence, lowering the operating frequency of the cores should not impact the overall performance of the application, while the energy consumption can be reduced.
The goal of this set of benchmarks was to obtain a power consumption pattern bound to the problem size and CPU frequency, and figure out whether a trade-off between frequency, performance and power consumption can be observed, thus highlighting a more efficient approach from the energy point of view.

3.3.1. Frequency and problem size scaling

HPCG shows that there's a trade-off point, where rising the frequency is not convenient anymore.

The walltime in this case is almost constant, as it is imposed as an input parameter.

The performance obtained with increased frequencies doesn't change much, and it doesn't change at all in some cases, as the CPU cannot be fed with new instructions until the data needed for the computation is retrieved from the memory, fact that bottlenecks the application.
Running HPCG at the lowest frequencies, gives a better performance-per-consumed-energy ratio. Increasing the frequency does not improve the performance, but increases a lot the power consumption without any benefit.

This means that running memory-bound applications at high frequencies is just a waste of energy, and from this point of view, may be more convenient lowering the frequency of the CPU when this kind of applications are supposed to be run. This, of course, is a matter of study for resource management software.

The plots in Appendix E.4. exhibit the results obtained on the run with the various sizes mentioned above, for all the frequency steps. The imposed runtime is 300 seconds. Should be noted that, excluding the runtimes shorter than 60 seconds which presented various anomalies, especially with the larger sizes (probably due to allocation/initialization issues), the results with runtime above 90 seconds reported consistent results. Only the size 64 (Figure E:35) is reported in this section as Figure 3.3.1:1.

The plots denoted a clear trend. Increasing the frequency leads to better performance, but leads to larger power consumption at the same time. From the energy-efficiency point of view, there is no benefit in running HPCG at the highest frequencies. The best compromise seems to be in the range 1.5 - 2 GHz. It is also evident that the best performing results are obtained with the Turbo Boost enabled, but they are affected by the largest power consumption, making this combination less efficient than running at the minimum frequency, 1.2 GHz. In case the energy-to-solution has the precedence over the time-to-solution, the lower frequencies represent the most convenient choice.

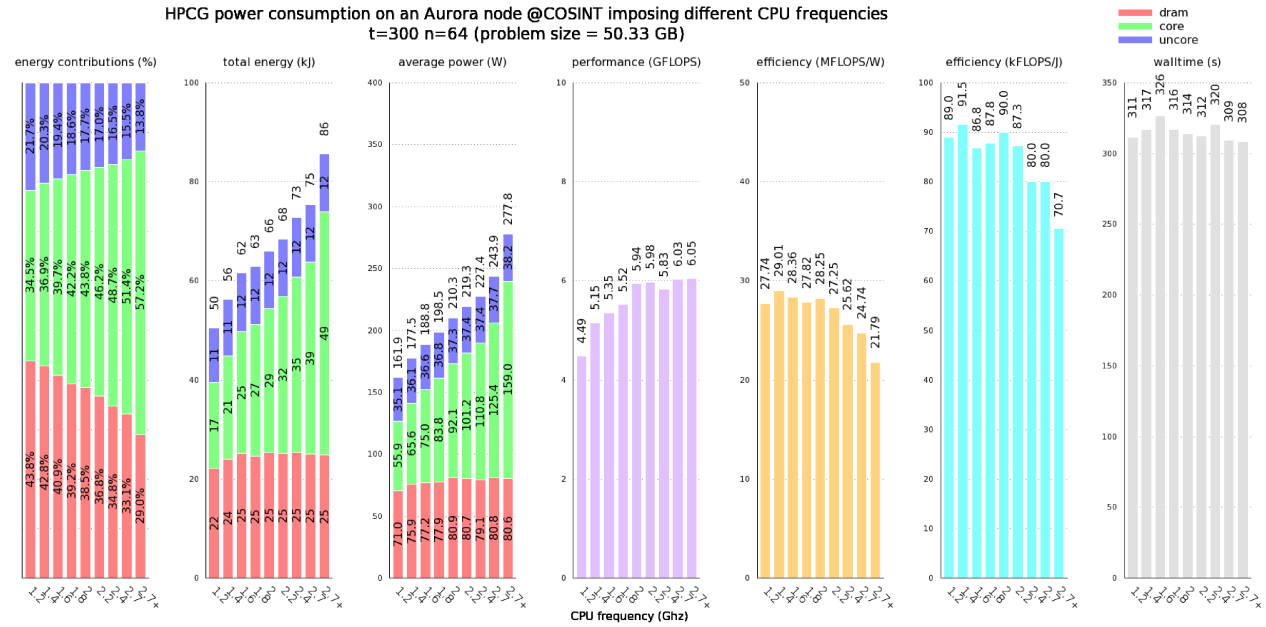


Figure 3.3.1:1: HPCG: problem size scaling vs. frequency scaling (size=64)

3.3.2. Top-down characterization

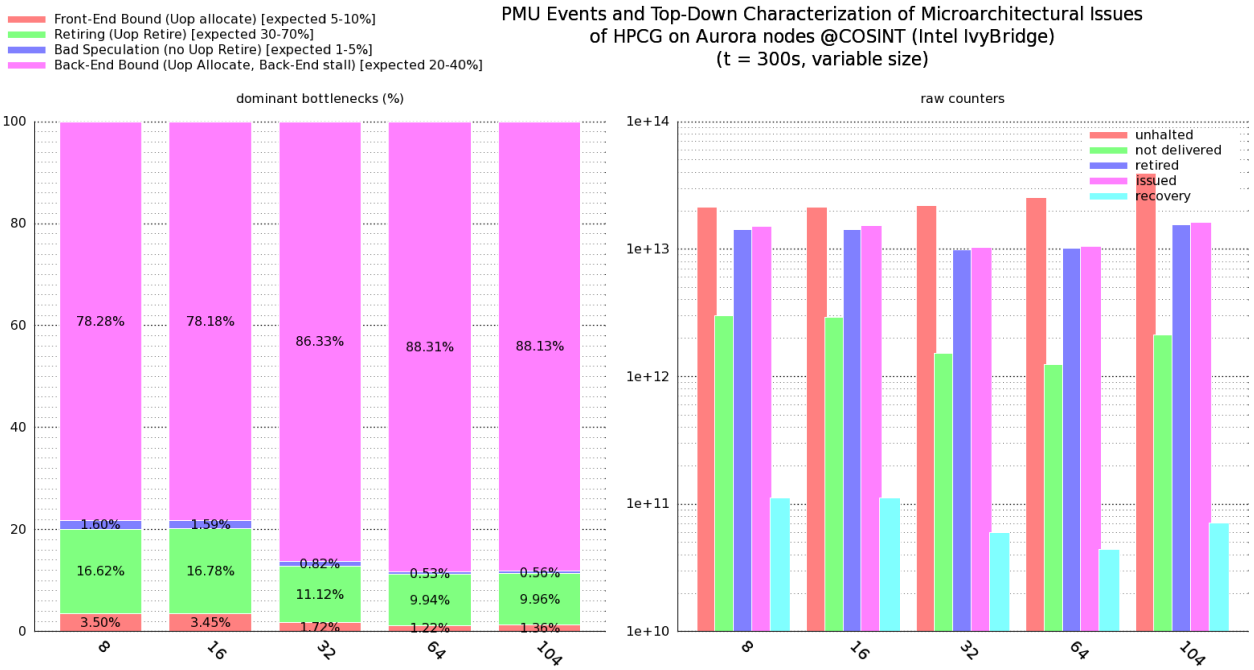


Figure 3.3.2:1: HPCG: TMAM

Figure 3.3.2:1 reports the various sizes for the 300 seconds run. The plot clearly shows that HPCG is a back-end bound application. As already stated before, this means that there might be an issue with memory accesses (LLC misses), which will be confirmed by the performance counters analysis. For small sizes, where the problem fits better in cache and memory accesses are less frequently needed, the application denotes a smaller dependency on the back-end, but it is still outside the optimal range nonetheless.

3.3.3. Performance counters

The analysis of the basic performance counters confirms that HPCG presents a large cache-miss ratio, and that for small sizes, this ratio is lower, as the required data can be found a little more often in cache.

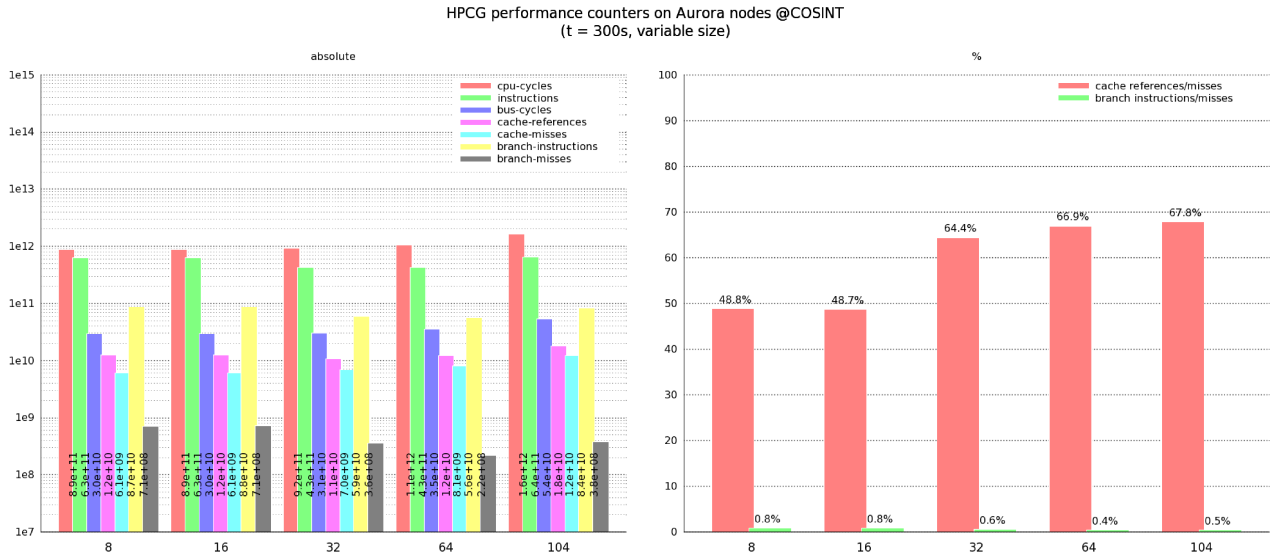


Figure 3.3.3:1: HPCG: Performance counters

3.3.4. Summary of HPCG investigation

The results of the HPCG investigation can be summarized as follow:

- although HPCG is a memory-bound application, higher CPU operating frequencies still play an important role on the time-to-solution and the performance obtained by the benchmark;
- when the energy-to-solution becomes a predominant factor (e.g. under power capping policies), HPCG results more energy-efficient at lower frequencies;
- the minimum frequency supported (1.2 GHz), is far more energy-efficient than Turbo Boost; this notable result can be used to tune/implement careful energy aware policies on HPC infrastructure;
- problem size scaling doesn't seem to affect the energy consumption of HPCG, nonetheless, the results shown by TMAM analysis and the performance counters analysis constitutes a basis for the comparison with other applications.

3.4. Quantum ESPRESSO

For real-world applications, identifying a trade-off for the energy efficiency is a non-trivial task. The performance (FLOPS) is usually not reported, hence the efficiency as defined before (FLOPS/W, FLOPS/J) cannot be easily computed. The performance counters allows to roughly estimate the number of floating point operations executed during the run of a monitored application. It was observed, by comparing the performance reported by HPL and other trivial matrix-multiplication codes, that the FLOPS count obtained by using the performance counters is smaller. Consistently on these tests, it appeared that measured FLOPS were a factor 1.2 smaller than what reported by the applications, although online documentation often reports over-counting associated to this technique.[82][83][75][77][78][79][80][81]

The performance measure reported in the following plots, was obtained by applying the results of these observations.

In order to exclude the effect of fluctuations, each test was repeated 10 times and the average is reported.

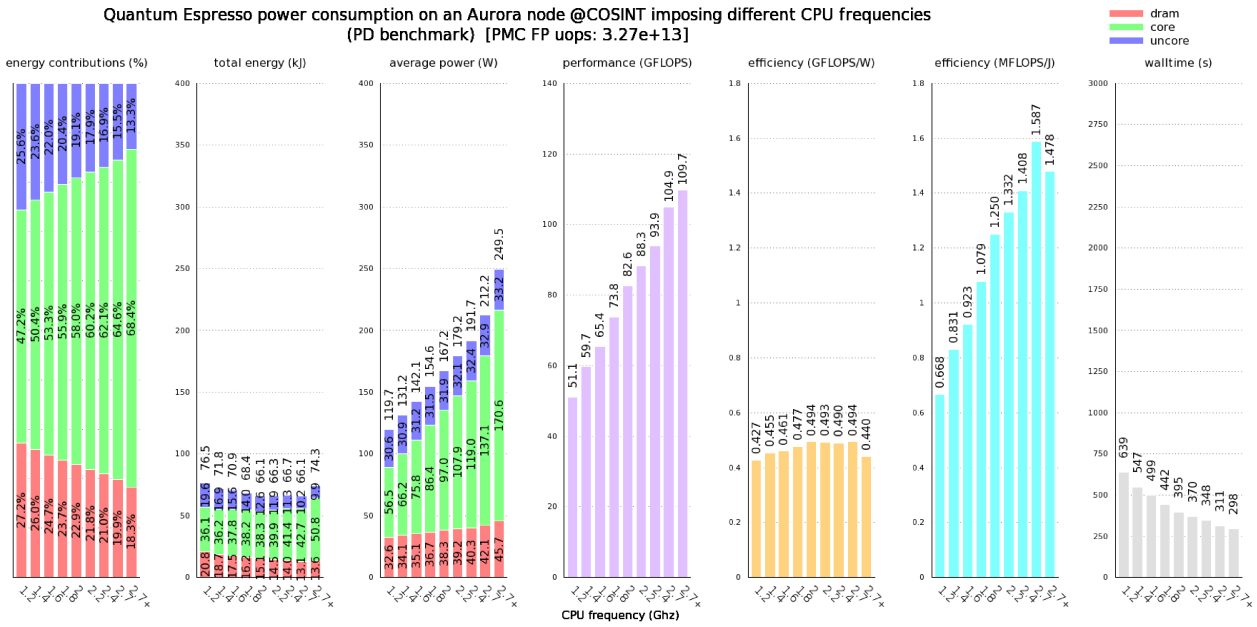


Figure 3.4:1: Quantum ESPRESSO, (scaled down) Palladium benchmark vs. frequency scaling

Figure 3.4:1 shows that shorter time-to-solution can be achieved by increasing the frequency. At the same time, the lower average power obtained with the lower frequencies allow to keep almost constant the energy-to-solution, thus highlighting how this benchmark benefits from higher computing speed.

The top-down analysis shows that this benchmark is a borderline case, concerning the optimal values, placed in the middle of what was observer from HPCG and HPL. Even though the larger FE-bound and retiring slots are still within the optimal range, BE-bound slots is close to the limit. Within the limits expressed by HPL and HPCG, this kind of pattern is often expected to be found for standard applications.

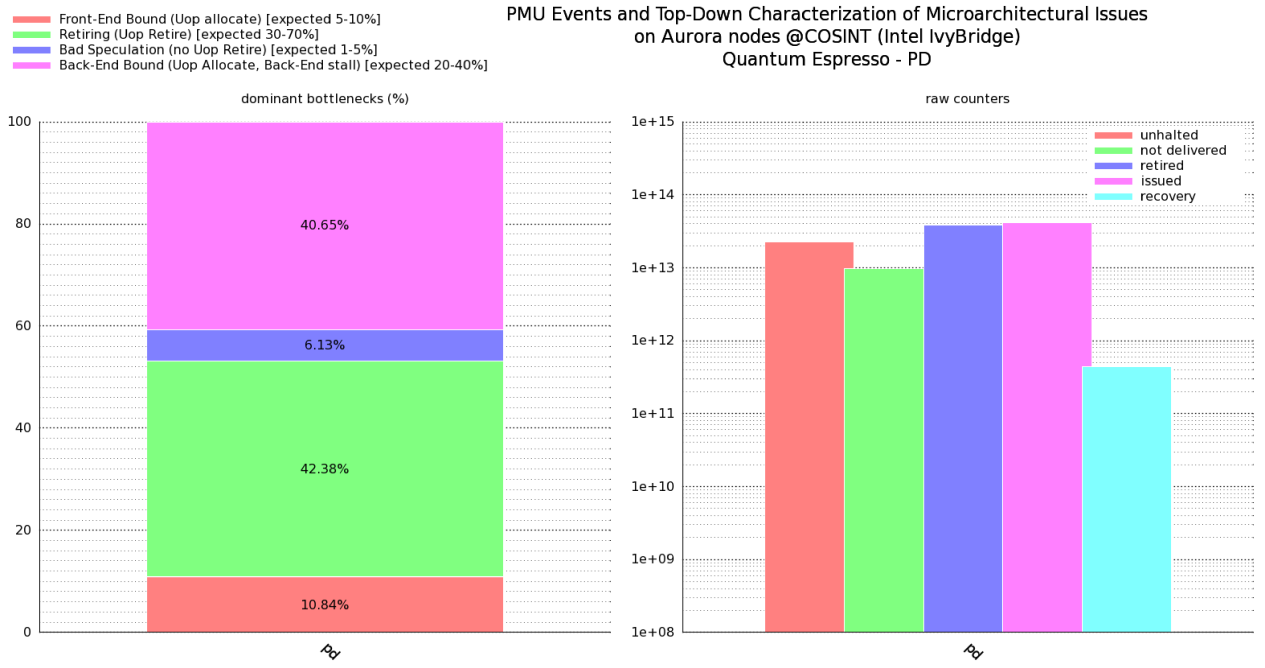


Figure 3.4:2: QE: TMAM

The performance counters analysis doesn't show anything relevant, relatively small cache-miss ratio compared to HPCG, and a slightly higher branch misprediction.

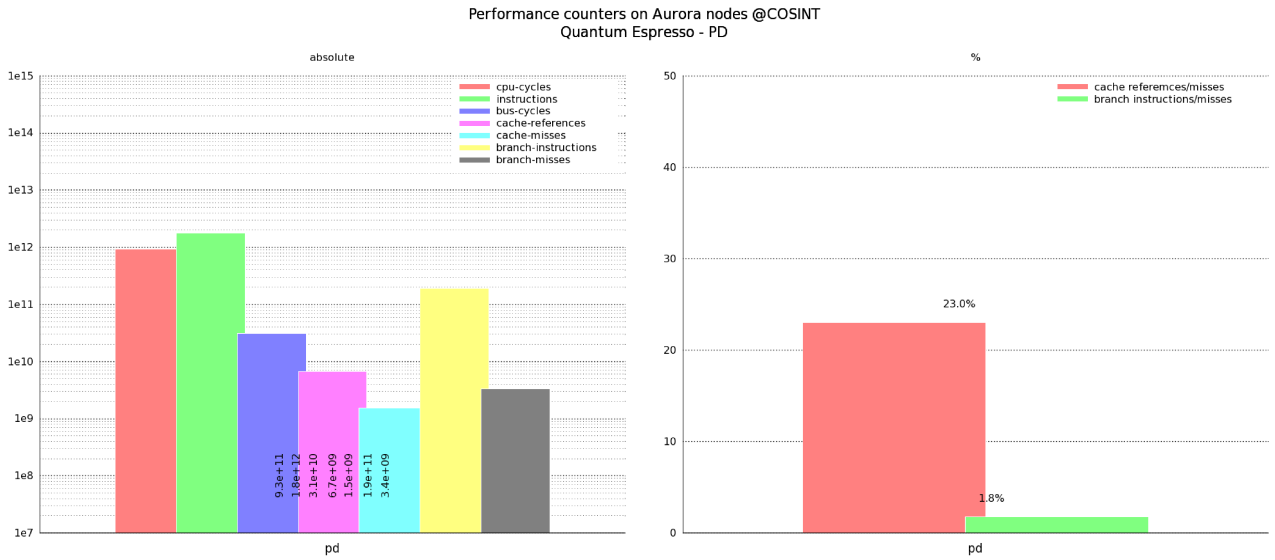


Figure 3.4:3: QE: Performance counters

3.5. LAMMPS

In this subsection, LAMMPS is tested using the Lennard-Jones liquid benchmark. Both frequency scaling (Figure 3.5:1) and problem size scaling (Figure 3.5:2) have been performed. Even though the same input was used for all these tests and the same random seed was used to initialize the data, some fluctuations were observed and therefore the reported results represent the average of 10 runs.

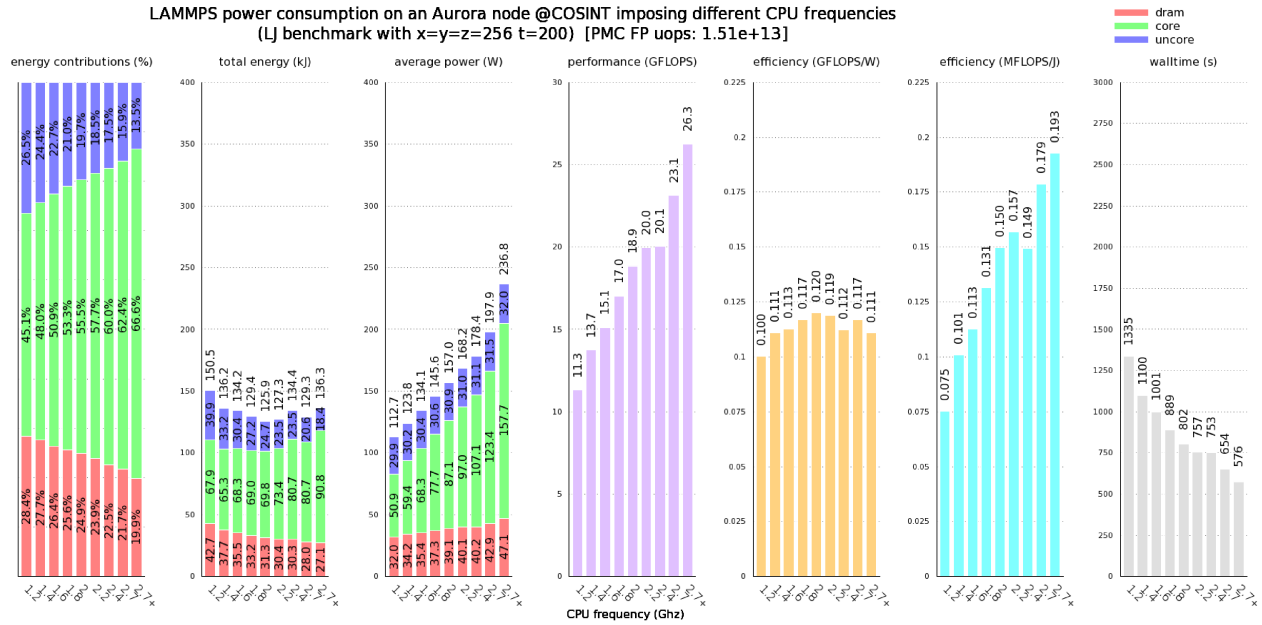


Figure 3.5:1: LAMMPS: frequency scaling

Figure 3.5:1, reporting the frequency scaling, depicts an energy-to-solution with small fluctuations, showing that running this benchmark at low frequencies won't be advantageous neither for the overall energy consumption nor for the time-to-solution. Running at the nominal frequency, though, seems to be more convenient than having the Turbo Boost enabled.

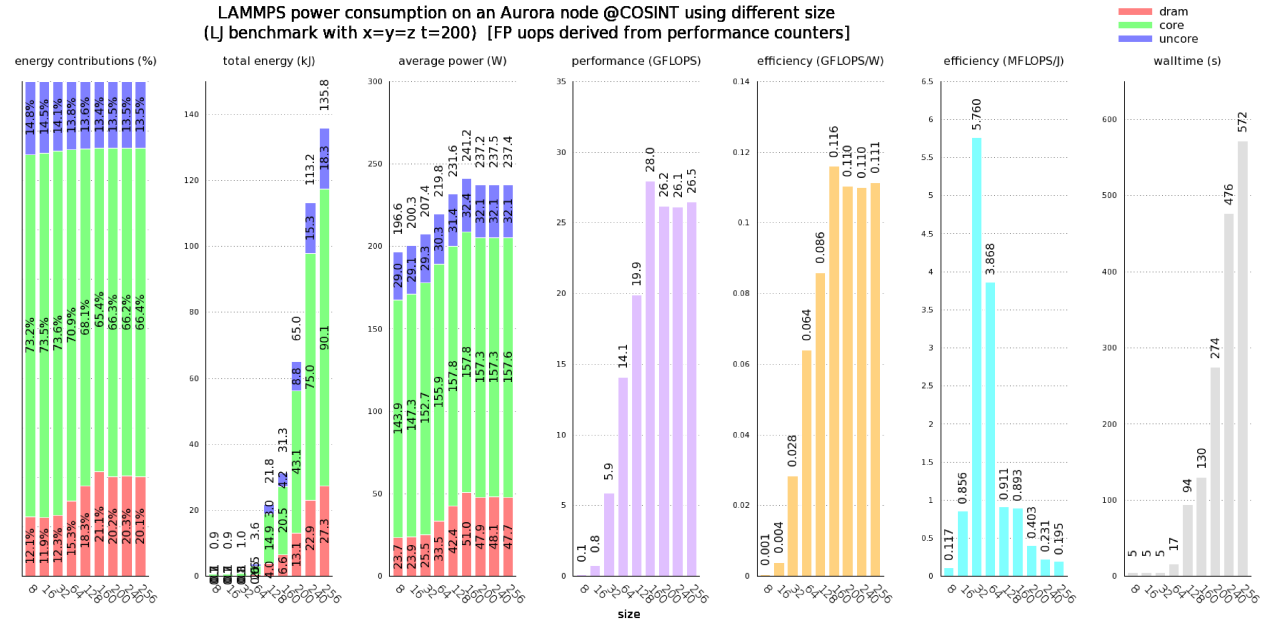


Figure 3.5:2: LAMMPS: problem size scaling

The problem size scaling, Figure 3.5:2, shows a large difference in the DRAM consumption. Small sizes fit in cache, hence increasing the CORE consumption while decreasing DRAM accesses and the power required. At a certain point in size, the data are becoming too large to fit in cache, and the pattern shows the DRAM power rising again.

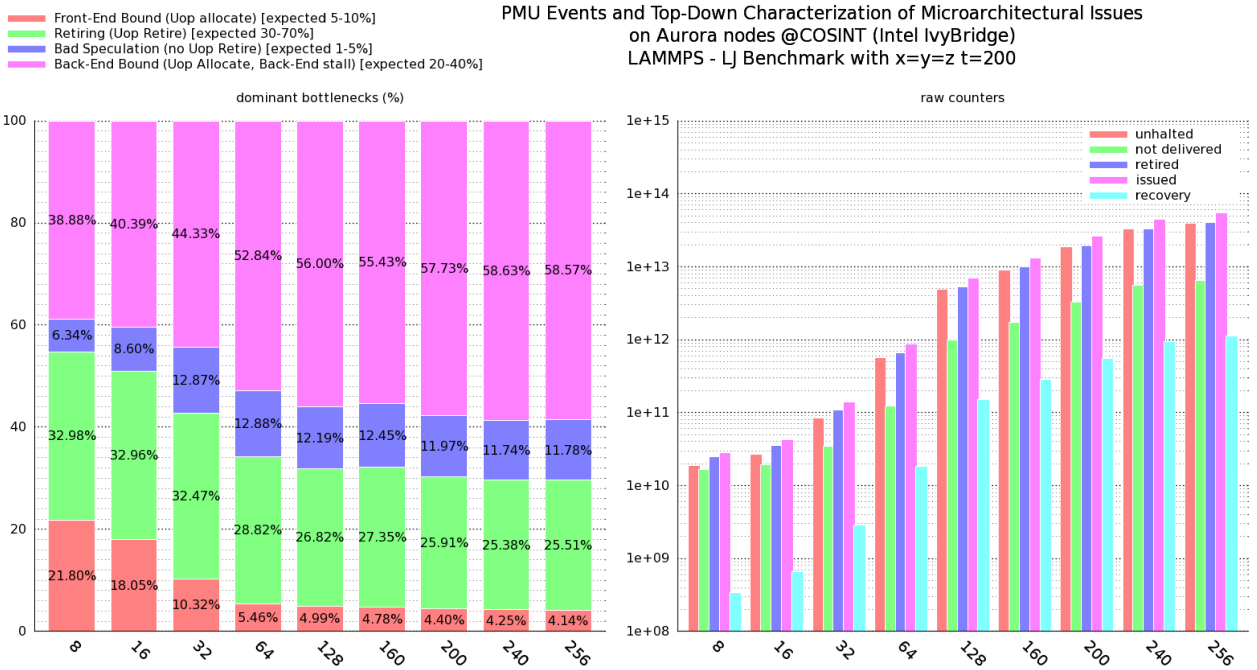


Figure 3.5:3: LAMMPS: TMAM `mpirun-perf`*, problem size scaling*

Figure 3.5:3 exhibits the expected behavior in respect of the problem size scaling. For very small sizes, the application is FE-bound, memory accesses are directed towards the cache, and the instructions cannot be fed fast enough to fill the pipeline. Unlike any other benchmark examined before, bad-speculation slots are larger than in any other test. This is likely due to the algorithm. The performance counters analysis should evidence this behavior through the branch misprediction ratio.

Increasing the size, again the application becomes memory-bound, although the bad-speculation is not affected.

For this benchmark, a further rough analysis of the impact of `mpirun` was performed on the top-down characterization and the performance counters analysis by using two combinations of commands:

- `perf stat -a ... mpirun -np 24 ...` (a single `perf` instance collects counters from all cores)
- `mpirun -np 24 ... perf stat ...` (`mpirun` spawns 24 `perf` instances, each collecting counters for one process bound to one core)

The latter, is the method used for most of the benchmarks presented so far.

The Figure 3.5:4, shows the effects that `mpirun` has on the TMAM considering both the benchmark program and `mpirun` altogether. By comparing Figure 3.5:3 and Figure 3.5:4, it can be noted that although for the smallest size 10% of the slots are moved from being BE-bound to FE-bound, no particular impact is evident. Hence, for future tests, changing the order of `mpirun`/`perf` execution shouldn't relevantly affect the collected readings.

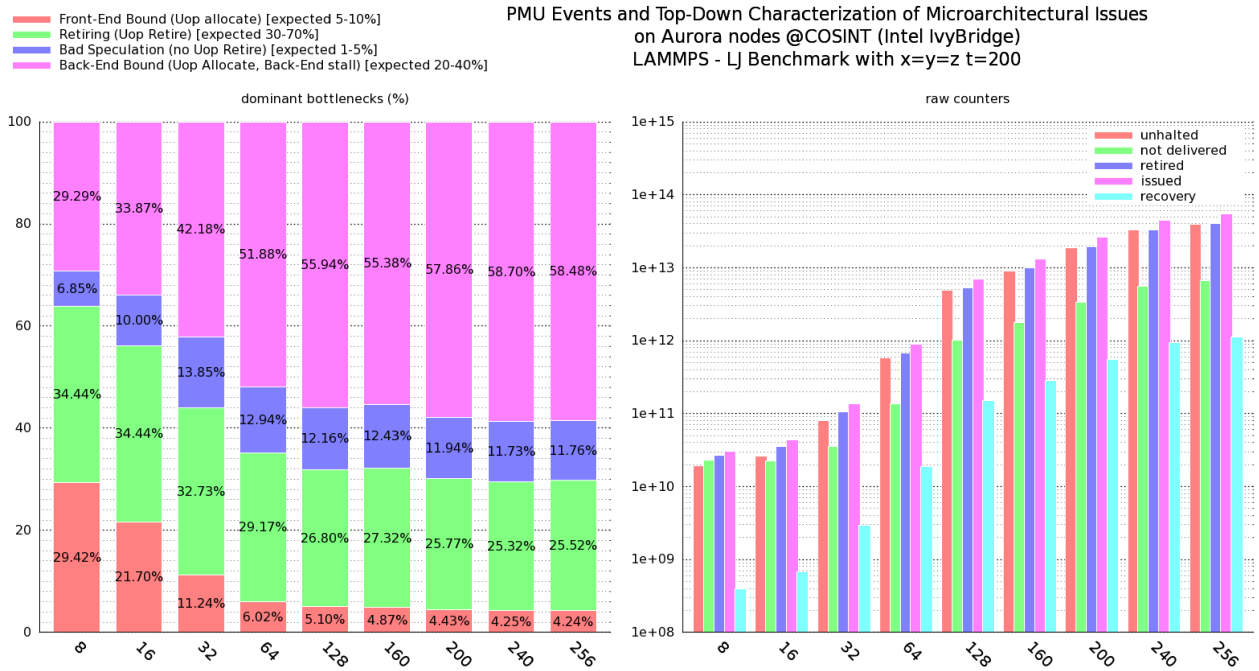


Figure 3.5:4: LAMMPS: TMAM `perf-mpirun`*, problem size scaling*

As expected, figure Figure 3.5:5 shows that a relatively large branch misprediction affects the benchmarks. The cache-miss ratio becomes relevant at size 160, where almost half the cache accesses don't get satisfied by a hit. For the other sizes, cache-misses are still relevant, making these results closer to the one obtained by HPCG than HPL's, showing again how HPL is far from being representative of real-world applications.

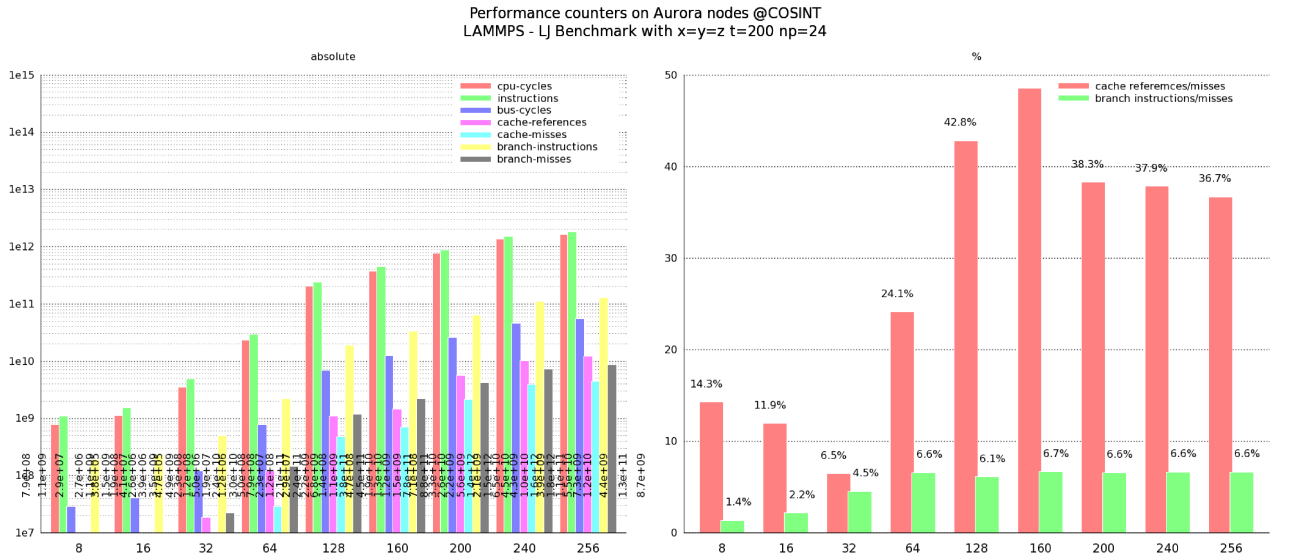


Figure 3.5:5: LAMMPS: `perf` *counters analysis,* `mpirun-perf`

Finally, Figure 3.5:6, compared to Figure 3.5:5, shows that `mpirun` doesn't seem to affect the results of analysis that include it together with the targeted application.

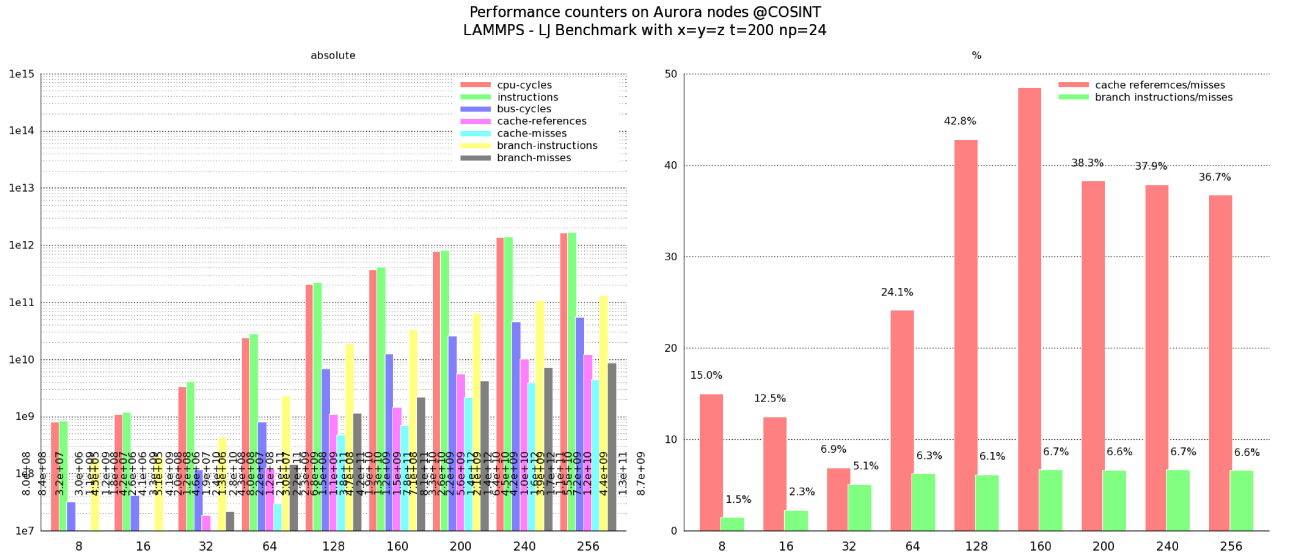


Figure 3.5:6: LAMMPS: `perf` *counters analysis,* `perf-mpirun`

3.6. Comparison

In this section, some selected results are compared and reviewed, for each of the three main analysis described in this document:

- energy efficiency vs. frequency scaling (3.6.1);
- top-down characterization (3.6.2);
- cache-miss and branch misprediction ratios using performance counters (3.6.3).

3.6.1. Energy efficiency and frequency scaling

The analysis performed on the frequency scaling behavior of the power consumption is summarized in Figure 3.6.1:1, which reports the energy efficiency, expressed as MFLOPS/J, in respect of the operating frequency of the CPUs.

Due to the large difference in efficiency, the results for each benchmark are presented on a different panel (1-5) with different scales for the Y axis, while the last panel on the right (6) compares the same numbers in log scale.

Must be noted that for HPL+Netlib, due to the extremely long runtime (>10 h), the only data available for the full frequency range was from the tests with 1/4 and 1/8 of the problem size (see Appendix E.3.). For the largest size, the energy-efficiency of HPL+Netlib should be 1 order of magnitude smaller (0.005 for the only full run with `ondemand` governor, reported in Figure 3.2.1:1).

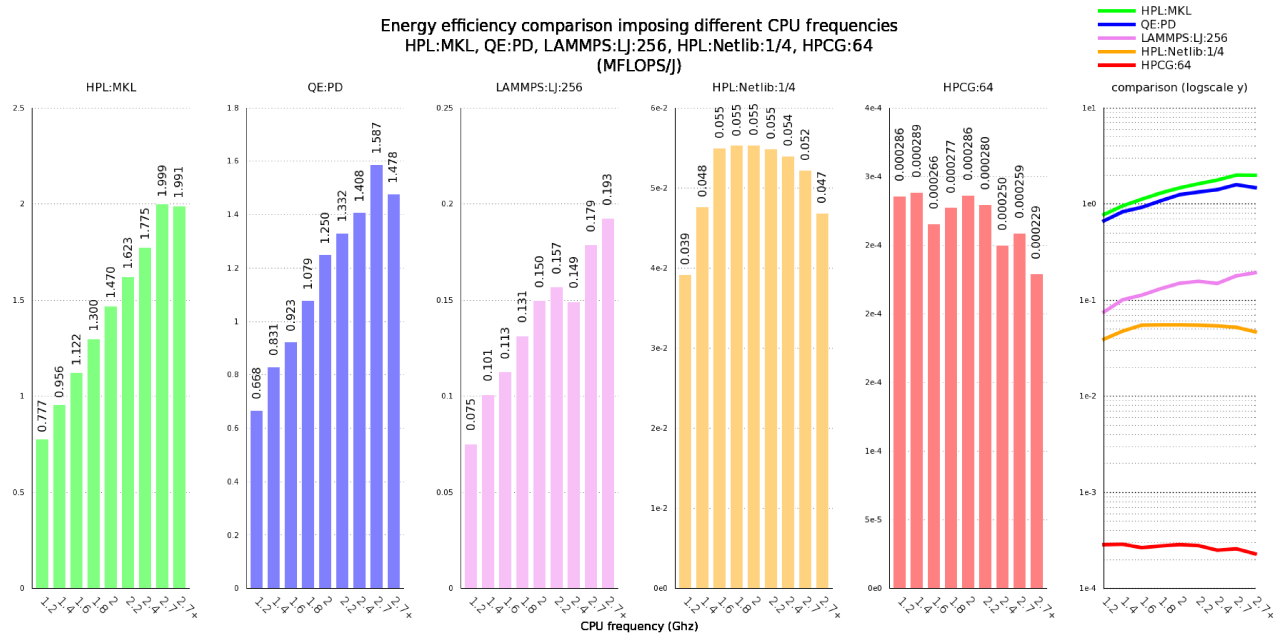


Figure 3.6.1:1: comparison: frequency scaling

From Figure 3.6.1:1 a clear pattern can be noticed. For CPU-bound applications (panels 1-3), the efficiency increases almost linearly with the frequency, as the shorter time-to-solution makes irrelevant the difference in average power consumption given by running at lower frequencies. For

memory-bound applications (panels 4-5), the trend is almost the opposite, as the CPUs cannot compute enough data to exploit the additional performance provided by the higher frequencies.
As can be seen from the comparison on panel 6, the efficiency spans over 4 orders of magnitude. The cost of having a memory-bound application, or an unoptimized application that could have been tuned or been written better to be more CPU-dense, is therefore proportional.

This final analysis shown that real-world applications are not well represented by HPL, and therefore tuning hardware specifications and choosing architectural features exclusively on HPL performance, may not be necessarily wise. HPL still represents the most that a system can possibly achieve. Unfortunately, HPCG does not represent the worst that can be achieved.

3.6.2. Top-down characterization

This section collects the most significant results obtained from the top-down characterization of HPL+MKL, Quantum ESPRESSO, LAMMPS, HPL+Netlib and HPCG. These tests were conducted with the `ondemand` governor and Turbo Boost enabled. The raw counters were collected using `perf`, and the data post-processed by means of `gnuplot` by using the formula described in Appendix B.

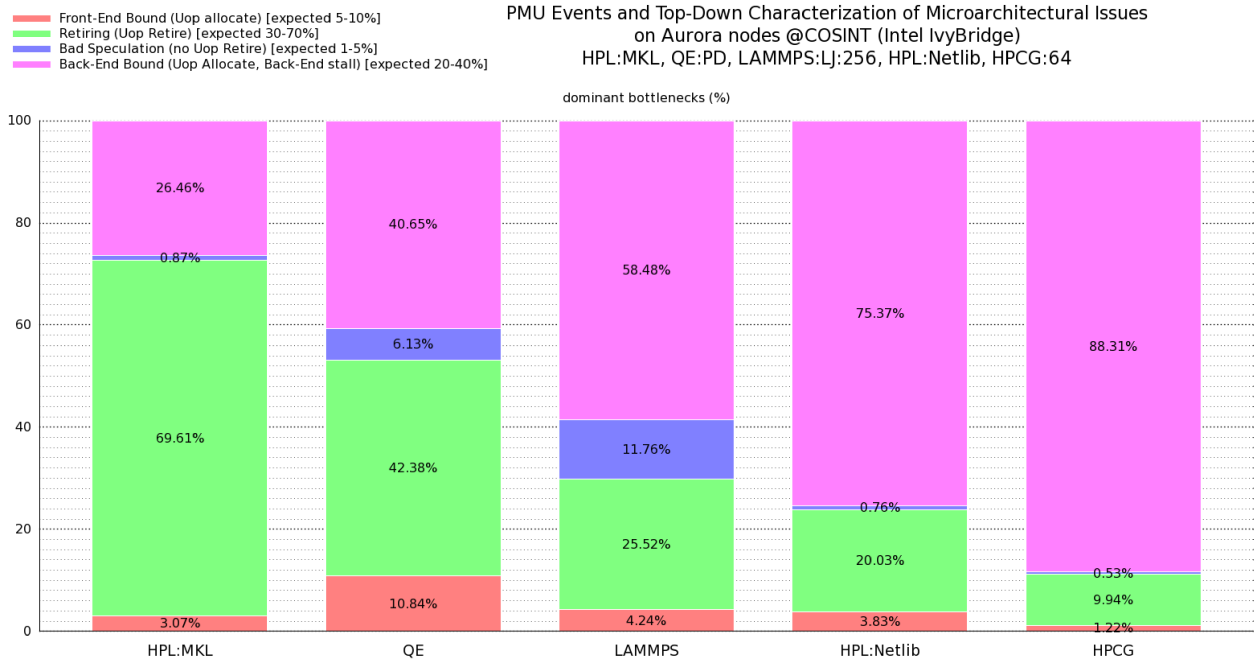


Figure 3.6.2:1: comparison: TMAM

In Figure 3.6.2:1, not surprisingly, the applications known to be memory-bound appear to be back-end bound. Surprisingly, though, HPL+Netlib behaves the opposite of HPL linked to any other library, actually closer to HPCG, explicitly designed to be memory-bound.
It is also evident that real-world applications, with complex algorithms and implementations, present a larger waste of resources due to bad speculation, while the benchmarks initially tested didn't expose any influence derived from branch prediction.

Besides the neat profiling and optimization aid that this procedure provides, this kind of analysis shown also that a “black-box” program can be categorized as CPU or memory-bound application in a nonintrusive and unattended way, just by wrapping the application and collecting performance counters out-of-band. In a vision of building an internal database for an energy-aware scheduler, this kind of information may be considered as an additional attribute, as long as it can be related to application *and* input file or memory utilization. As it was evident from the problem size scaling analysis, the very same application with a different problem size or input file may give completely different results concerning both performance and power consumption. The reliability of this attribute would be therefore debatable.

3.6.3. Performance counters

Figure 3.6.3:1 reports a comparison of the cache-miss and branch misprediction ratios of HPL+MKL, Quantum ESPRESSO, LAMMPS, HPL+Netlib and HPCG. As for the top-down characterization, these tests were conducted with the `ondemand` governor and Turbo Boost enabled.

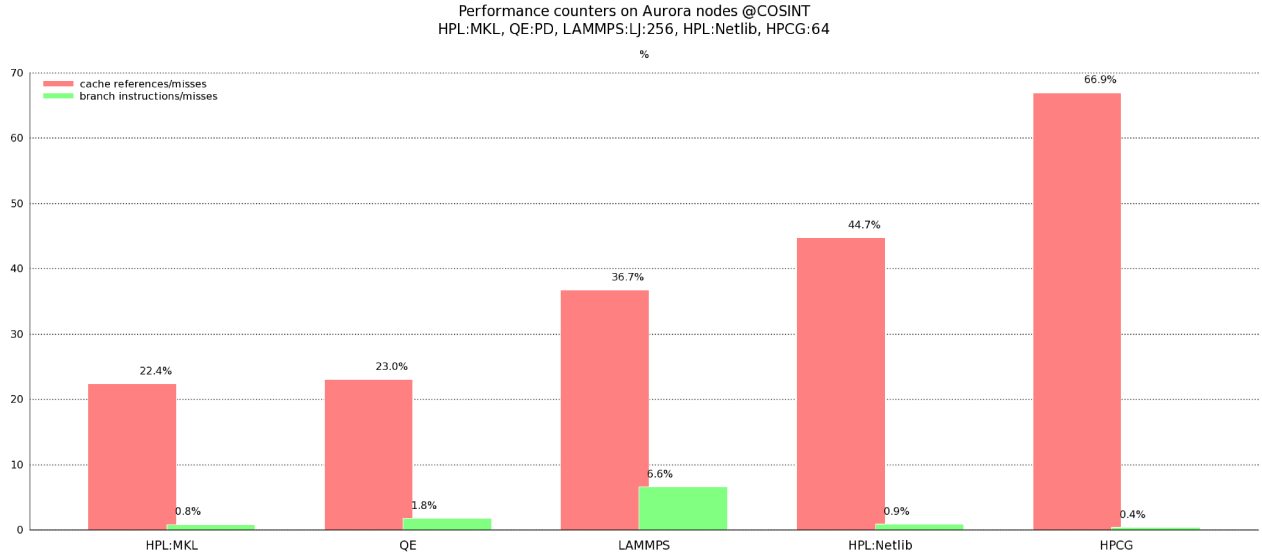


Figure 3.6.3:1: comparison: performance counters

Once again, the trend is clear and doesn't need further comment, but something must be noted. For an application, being dependent on large amounts of memory does not necessarily relate to being memory-bound too. HPL, even though it is using more than 80% of the available memory in these tests, can be extremely efficient in fetching the data, as the algorithm allows to exploit the cache levels and data locality. Of course, many algorithms cannot be further optimized and the problem of sparse data cannot be efficiently addressed, but for sure, a large amount of programs still can be optimized to take advantage of the new architectures and the new features, increasing drastically both the performance and the energy efficiency. The performance counters proved to be an important tool in this respect, as a simple analysis can give lots of hidden information and deep insights concerning a program that might not be noticed by reading, or writing, the code.

4. Conclusions

This section briefly summarizes the activity performed in this research project, highlights the results obtained and the lessons learned. It also addresses some possible future directions and ideas that could be further investigated.

A careful and in-depth performance/energy analysis has been conducted on a set of HPC workloads: two standard and well known benchmarks and two scientific application widely used.
Several important results have been achieved:

- It was evidenced how HPL and HPCG currently represent two extremes of HPC applications. One largely CPU-bound, one memory-bound.
- Other benchmarks shown that real-world applications are in between these two border cases. QE and LAMMPS, at least for the input data and benchmarks tested, appeared as 2 typical applications, with complex algorithms, lying within the range described by HPC and HPCG.
- HPL was tested and studied using different libraries and it was clearly shown that both performance and energy impact can be largely affected by the kind of library. Netlib, even though considered the reference BLAS and LAPACK implementation, if used for HPC applications can lead to bad performance and wasted resources, while other libraries (not necessarily commercial implementations) could provide better and more efficient alternatives.

This study, despite the limited amount of time used, provides some clear indications about what can be done and how it can be used for energy-efficiency optimization.

It was observed that the energy-efficiency of memory-bound applications actually benefits from running at lower CPU frequency. Deciding to downclock an application in order to spare few watts, though, is not always feasible nor useful.
Nevertheless, under a regime of power capping, knowing how an application behaves and reacts may be useful to understand whether to run the application anytime, with a lower clock frequency limiting the overall power consumption, or wait for available resources and run it at the fastest speed possible for a shorter time. It was evident that what largely influences the energy-efficiency is the time-to-solution more than the instant power consumption, as a greater performance for a shorter time may be more convenient than a lower performance for a longer time.

On the technical side it could be noted the following:

- RAPL turned out to be a great feature, allowing to monitor the power consumption of a system. Maybe not really meaningful for a single computer, in a large infrastructure and coupled to a resource manager it can provide deep and accurate insight about how hardware and software behave and interact.
- Performance counters represent a huge resource for code profiling and optimization. Many caveats are hidden in this area, and often this doesn't immediately appear at first sight, often taking for granted what standard tools report.

What was exposed in this work is just the tip of an iceberg, what lies underwater is impressive. For example, an attempt (not fully documented) was made to figure out the best performance counters in order to obtain L1i/L1d/L2/L3/TLB. Despite the efforts, it was extremely difficult to figure out whether a measure was reliable or not. Most counters offer speculative counts, not real hardware counts. Some include or exclude particular corner cases, or some specific metrics. Often, the

available documentation is not detailed, or a particular feature is undocumented or kept hidden.
Even the same performance events are reported differently depending on the microarchitecture, and the documentation often doesn't keep up with these changes.
The potential is therefore great, and in this thesis we just started exploiting it.

Moreover this project have produced several tools that could have a significant impact on the efficient energy management of an HPC infrastructure.
Here are the most important achievements:

- The monitoring software was improved (`hwloc`, `numactl`, `sysconf`, command line options).
- New `pductl` utility for remote command line based control and monitoring of the PDU.
- All the tools used were made available on HPC cluster for further study, many also as loadable modules.
- `LIKWID` and `pmu-tools` data structures have been interfaced to `perf`, allowing greater flexibility.
- The platform was benchmarked with greater results than those officially given.

All these tools are planned to be made available soon in a public git repository as open-source software.

4.1. Future Perspectives

This investigation opened the Pandora's box. A huge amount of data was collected and analyzed in many aspects. Some of the metrics obtained and a large portion of data had to be ignored, and many other metrics could have been used for completely different analysis.
The performance counters can be used for a huge variety of different microbenchmarks and deeper analysis of software and hardware behavior. For instance, in order to estimate the number of floating-point operations per seconds of QE and LAMMPS, many counters have been collected. These counters, could have led to a study about how an application uses or not vectorization.

This study unveiled a wide range of opportunities for further investigation. Among these, some have been already scheduled in the near future.

1. Lots of efforts have been spent on understanding performance counters, their meaning, identifying the most useful ones, the (un)availability and/or (un)reliability of many of them depending on the architecture, and especially how different tools and libraries handle them. A deeper analysis may expose performance events not only useful for code profiling and optimization, but also for providing metrics to be integrated into resource management systems, schedulers, and monitoring tools. Intel's official profiling tool, VTune[84], wasn't used in this study, but could provide some useful insights about official Intel's strategies on handling performance monitoring events.

2. Due to time constraints, the analysis of GPU power consumption was only explored enough to give a hint about what could be the influence of moving the calculation from the CPU to the accelerator, GPU DVFS wasn't approached at all. Exploring means to measure the power

consumption on MIC and GPGPU is therefore deemed important for the forthcoming generations of “accelerated” codes. Moreover, performance counters and frequency scaling are planned to be investigated also on these platforms.

3. The current evolution of HPC technologies (and market trends) are making of energy-efficiency the new hot-topic. Large infrastructures requires fine-grained monitoring of the resources, and as well, a wiser and more energy-friendly approach to resource management. Integration of energy profiling features and power capping policies in queue managers and schedulers is rapidly evolving. The experience gained during this work offers a new opportunity to enhance these features, and to apply similar methodologies to a production environment in order to deepen the insights already acquired with real-world usage.

4. This project focused on exploratory testing, and challenged advanced and innovative methods of investigation of application-level performance and energy-efficiency. In this work some representative applications were benchmarked, therefore studying a broader spectrum of applications may lead to even more relevant insights. Testing different scientific workload and different real-world applications will allow to create, in an unattended or semiautomatic way, a database of energy efficiency for HPC clusters/infrastructures.

5. An attempt was made to gather external power readings from various devices. Even though it didn't provide accurate nor usable results on the tested platform, this strategy still represents a good approach for other infrastructures. Many other devices available on HPC systems may provide useful information for a wider and more accurate analysis of the system (network, storage, fans, power supplies, motherboard and overall node power consumption), and multinode scaling analysis may identify the impact of network-based communication on the overall energy consumption of an application. A study on this respect will be also evaluated.

5. Acknowledgments

I would like to express my gratitude to all the people who made this project possible.

I wish to thank COSINT for providing the infrastructure on which this research project was based.

I appreciate the support from CNR-IOM DEMOCRITOS that allowed me to participate to the MHPC and granted me the time to conduct the research discussed in this dissertation.

My deepest gratitude to Dr. Andrea Bartolini (Micrel Lab) and Dr. Stefano Cozzini (MHPC), my research supervisors, for their advice and assistance in keeping my progress on schedule.

I am grateful to all the teachers of the Master, for their patient guidance, enthusiastic encouragement and useful critiques. I would like to acknowledge Dr. Christopher Dahnken (Intel, MHPC), whose lectures inspired the performance counters analysis discussed in this dissertation.

Assistance provided throughout the Master by Giuseppe Piero Brandino, a.k.a. Pino, was greatly appreciated.

I wish to extend my greatest and deepest appreciation and infinite gratitude to all the FOSS software developers, system administrators and technical forums contributors all over the world, for the huge amount and variety of software and helpful documentation made available on the web, making very complex and extremely obscure tasks few clicks away from anyone's knowledge.

Thanks to Intel, for the great manuals and documentation, but please, next time make it grep'able and parsable from the command line.

I also wish to thank the fellow students of the MHPC 2014-2015, for the help provided and for keeping me sane through all the difficulties of this intense year.

Finally, I would like to express my heart-felt gratitude to Giorgia, for her support, understanding and patience.

I would also like to apologize to all mankind and planet Earth for the huge amount of resources wasted to accomplish this research project on... well... energy-efficiency. Sorry.

6. Bibliography

1: Eurotech, Aurora Systems, http://www.eurotech.com/en/hpc/hpc+solutions/data+center+hpc/Aurora+Systems

2: CO.S.IN.T., Consorzio per lo Sviluppo Industriale di Tolmezzo, http://www.cosint.it

3: Eurora, EURopean many integrated cORe Architecture, http://www.top500.org/system/178077

4: CINECA, http://www.cineca.it/

5: Green500, 2013/06, http://www.green500.org/lists/green201306

6: Top500, http://www.top500.org/

7: Thiane2, http://www.top500.org/system/177999

8: QUARTETTO, http://www.top500.org/system/178251

9: K. Bergman, et al., Exascale computing study: Technology challenges in achieving exascale systems., 2008, http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf

10: Green500, http://www.green500.org/

11: Shoubu, http://www.top500.org/system/178542

12: Borghesi A., Conficoni C., Lombardi M. and Bartolini A., MS3: A Mediterranean-style job scheduler for supercomputers - do less when it's too hot!, International Conference on High Performance Computing & Simulation (HPCS), 2015, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7237025

13: Srinivas Pandruvada, Running Average Power Limit - RAPL, https://01.org/blogs/tlcounts/2014/running-average-power-limit-%E2%80%93-rapl

14: Netlib, http://www.netlib.org/blas/

15: ATLAS, http://math-atlas.sourceforge.net/

16: OpenBLAS, http://www.openblas.net/

17: PLASMA, http://icl.cs.utk.edu/plasma/

18: MKL, https://software.intel.com/en-us/intel-mkl

19: QE, http://www.quantum-espresso.org/

20: LAMMPS, http://lammps.sandia.gov/

21: SLURM, http://slurm.schedmd.com/

22: Intel, Intel Xeon Processor E5-2697 v2 (30M Cache, 2.70 GHz), http://ark.intel.com/products/75283/Intel-Xeon-Processor-E5-2697-v2-30M-Cache-2_70-GHz

23: hwloc, Portable Hardware Locality, http://www.open-mpi.org/projects/hwloc/

24: CentOS, https://www.centos.org/

25: gcc, https://gcc.gnu.org/

26: OpenMPI, http://www.open-mpi.org/

27: PBSPro, PBS-Pro, http://www.pbsworks.com/

28: Yiannis Georgiou, David Glesser, Krzysztof Rzadca and Denis Trystram, Introducing Energy based fair-share scheduling, 2014, http://slurm.schedmd.com/SUG14/energetic_fair_share.pdf

29: Yiannis Georgiou, Enhancing Slurm with Energy Consumption Monitoring and Control Features, 2012, http://slurm.schedmd.com/slurm_ug_2012/Energy_Accounting-BULL-SUG2012.pdf

30: Bartolini, Andrea, et al., Unveiling eurora-thermal and power characterization of the most energy-efficient supercomputer in the world., Proceedings of the conference on Design, Automation & Test in Europe. European Design and Automation Association, 2014, http://www.date-conference.com/files/proceedings/2014/pdffiles/10.3_2.pdf

31: Micrel, Micrel Lab, http://www-micrel.deis.unibo.it/sitonew/

32: UNIBO, Università di Bologna, http://www.unibo.it/

33: numactl, numactl/libnuma, http://oss.sgi.com/projects/libnuma/

34: LIKWID, https://code.google.com/p/likwid/

35: NVML, NVidia Management Library, https://developer.nvidia.com/nvidia-management-library-nvml

36: nvidia-smi, NVIDIA System Management Interface, https://developer.nvidia.com/nvidia-system-management-interface

37: snmp-utils, Net-SNMP, http://www.net-snmp.org/

38: IPMItool, http://sourceforge.net/projects/ipmitool/

39: cpufrequtils, https://www.kernel.org/pub/linux/utils/kernel/cpufreq/

40: sysfs, https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

41: perf, https://perf.wiki.kernel.org/

42: Gnuplot, http://www.gnuplot.info/

43: HPL, http://www.netlib.org/benchmark/hpl/

44: HPCG, https://software.sandia.gov/hpcg/about.php

45: Jack Dongarra, Piotr Luszczek and Michael Heroux, Toward a new (another) metric for ranking High Performance Computing systems, 2014, http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140331/Dongarra.pdf

46: Jack Dongarra, ICSC 2014 Architecture-aware Algorithms and Software for Peta and Exascale Computing, 2014, http://icsc2014.sjtu.edu.cn/wp-content/uploads/2014/05/ICSC2014-Jack.pdf

47: Intel, "Intel Xeon Processor E5 v2 Product Family, 1/2, Datasheet - Volume One of Two", March 2014, http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-v2-datasheet-vol-1.pdf

48: CPUfreq, https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

49: Intel, "Intel 64 and IA-32 Architectures -- Software Developer's Manual -- Volume 3B: System Programming Guide, Part 2", September 2015, http://www.intel.it/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

50: Intel, Intel 64 and IA-32 Architectures Software Developer Manuals, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

51: Intel, "Intel 64 and IA-32 Architectures Optimization Reference Manual", September 2015, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

52: Intel, "Intel Xeon Processor E5 v2 and E7 v2 Product Families Uncore Performance Monitoring Reference Manual", February 2014, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/xeon-e5-2600-v2-uncore-manual.pdf

53: Intel, "Intel Xeon Processor E5 v2 Product Family, 2/2, Datasheet - Volume Two: Registers", March 2014, http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-v2-datasheet-vol-2.pdf

54: Intel, "Chapter 14, 14.9.2 POWER AND THERMAL MANAGEMENT: RAPL", September 2015, http://www.intel.it/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

55: Intel, "APPENDIX B: USING PERFORMANCE MONITORING EVENTS", September 2015, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

56: Alexander Supalov, Andrey Semin, Michael Klemm and Christopher Dahnken, Optimizing HPC Applications with Intel Cluster Tools: Hunting Petaflops, 2014,

57: Andi Kleen, Measuring workloads with toplev -- TopDown High level overview, https://github.com/andikleen/pmu-tools/wiki/toplev-manual

58: Intel, Appendix B: Using performance monitoring events -- B.1 Top-down analysis method, September 2015, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

59: Jackson Marusarz, How to Tune Applications Using a Top-Down Characterization of Microarchitectural Issues, https://software.intel.com/en-us/articles/how-to-tune-applications-using-a-top-down-characterization-of-microarchitectural-issues

60: Aman Singh and Anup Buchke, A Study of Performance Monitoring Unit, perf and perf_events subsystem, http://rts.lab.asu.edu/web_438/project_final/CSE_598_Performance_Monitoring_Unit.pdf

61: Treibig, J. and Hager, G. and Wellein, G., LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments, Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, 2010, http://arxiv.org/abs/1004.4431

62: libpfm, libpfm4/perfmon2, http://perfmon2.sourceforge.net/

63: pmu-tools, https://github.com/andikleen/pmu-tools

64: PAPI, http://icl.cs.utk.edu/papi/

65: msr-tools, https://01.org/msr-tools

66: perfmon, https://download.01.org/perfmon/

67: powertop, https://01.org/powertop

68: turbostat, http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/tools/power/x86/turbostat

69: cpupowerutils, http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/tools/power/cpupower

70: cpuspeed, http://carlthompson.net/Software/CPUSpeed

71: LAMMPS Lennard-Jones benchmark, http://lammps.sandia.gov/bench.html#lj

72: Linux Perf sources, user-space, https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/plain/tools/perf/

73: Linux Perf sources, kernel-space, https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/plain/arch/x86/kernel/cpu/

74: Intel, "[SDM3B2] Chapter 19, Performance Monitoring Events", September 2015, http://www.intel.it/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

75: Intel, "BT257. Performance Monitor Instructions Retired Event May Not Count Consistently", "Intel Xeon Processor E5 Product Family Specification Update", page(s) 87, January 2015, http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-e5-family-spec-update.pdf

76: Intel, Memory and Cache Profiling Erratum on Intel Xeon processor E5 family, https://software.intel.com/en-us/articles/performance-monitoring-on-intel-xeon-processor-e5-family

77: Intel, "Desktop 3rd Generation Intel CoreTM Processor Family Specification Update", April 2015, https://www-ssl.intel.com/content/dam/www/public/us/en/documents/specification-updates/3rd-gen-core-desktop-specification-update.pdf

78: Intel, "BV98. Performance Monitor Counters May Produce Incorrect Results", "Desktop 3rd Generation Intel CoreTM Processor Family Specification Update", page(s) 48, April 2015, https://www-ssl.intel.com/content/dam/www/public/us/en/documents/specification-updates/3rd-gen-core-desktop-specification-update.pdf

79: Intel, "BU1. Performance Monitor Instructions Retired Event May Not Count Consistently", "Desktop 3rd Generation Intel CoreTM Processor Family Specification Update", page(s) 52, April 2015, https://www-ssl.intel.com/content/dam/www/public/us/en/documents/specification-updates/3rd-gen-core-desktop-specification-update.pdf

80: Intel, "BV112. Performance Monitor Instructions Retired Event May Not Count Consistently", "Desktop 3rd Generation Intel CoreTM Processor Family Specification Update", page(s) 52, April 2015, https://www-ssl.intel.com/content/dam/www/public/us/en/documents/specification-updates/3rd-gen-core-desktop-specification-update.pdf

81: Linux Kernel Mailing List, https://lkml.org/lkml/2015/3/23/417

82: PAPI, Counting Floating Point Operations on Intel Sandy Bridge and Ivy Bridge, http://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:SandyFlops

83: Intel forum, Interpreting the AVX counter results, https://software.intel.com/en-us/forums/intel-vtune-amplifier-xe/topic/277877

84: VTune, https://software.intel.com/en-us/intel-vtune-amplifier-xe

85: Intel, Avoiding and Identifying False Sharing Among Threads, https://software.intel.com/en-us/articles/avoiding-and-identifying-false-sharing-among-threads

86: Jackson Marusarz, Cache Miss Rates in Intel VTune Amplifier XE, https://software.intel.com/en-us/articles/cache-miss-rates-in-intel-vtune-amplifier-xe

87: Peter Wang, Estimate the penalty of Cache Miss more accurate on Ivy-bridge?, https://software.intel.com/en-us/blogs/2013/07/01/estimate-the-penalty-of-cache-miss-more-accurate-on-ivy-bridge

88: Georgios Bitzes and Andrzej Nowak, The overhead of profiling using PMU hardware counters, 2014, http://openlab.web.cern.ch/sites/openlab.web.cern.ch/files/technical_documents/TheOverheadOfProfilingUsingPMUhardwareCounters.pdf

89: Intel, Intel 64 and IA-32 Architectures -- Software Developer's Manual -- Volume 1: Basic Architecture, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

90: Intel, Intel 64 and IA-32 Architectures -- Software Developer's Manual -- Volume 3 (3A, 3B, 3C & 3D): System Programming Guide, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

91: Intel, "Chapter 19, Performance Monitoring Events", September 2015, http://www.intel.it/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

92: Intel, "B.3.2: Locating Stalls in the Microarchitecture Pipeline", September 2015, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

93: Intel, "B.3.3.1: Precise Memory Access Events", September 2015, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

94: Intel, "B.4.2: Hierarchical Top-Down Performance Characterization Methodology and Locating Performance Bottlenecks", September 2015, http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

95: Intel, "Intel Xeon Processor E5 Product Family Specification Update", January 2015, http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-e5-family-spec-update.pdf

7. Appendices

Appendix A. List of Acronyms

AC: Alternating Current
ACPI: Advanced Configuration and Power Interface
ALU: Arithmetic Logic Unit
AMBER(MD): Assisted Model Building with Energy Refinement (Molecular Dynamics) package
API: Application Program Interface
APM: Advanced Power Management
ATLAS: Automatically Tuned Linear Algebra Software
AVX: Advanced Vector Extensions
BE: Back-End
BIOS: Basic Input-Output System
BLAS: Basic Linear Algebra Subprograms
CPI: Clock/Cycles Per Instruction
CPU: Central Processing Unit
CSR: Configuration Space Registers
DC: Direct Current
DMI: Direct Media Interface
DP: Double Precision
DRAM: Dynamic Random-Access Memory
DVFS: Dynamic Voltage and Frequency Scaling
EIST: Enhanced Intel SpeedStep
ESPRESSO: opEn-Source Package for Research in Electronic Structure, Simulation, and Optimization
ER: Embedded Controller
EURORA: EURopean many integrated cORe Architecture
FE: Front-End
FLOPS: Floating-Point Operations Per Second
FMA: Fused Multiply-Add
FOSS: Free and Open-Source Software
FP: Floating Point
FPU: Floating-Point Unit
GEMM: GEneral Matrix Multiplication
(GP)GPU: (General-Purpose) Graphics Processing Unit
HPC: High Performance Computing
HPCG: High Performance Conjugate Gradient
HPL: High Performance Linpack
IGP: Integrated Graphics Processor
IMC: Integrated Memory Controller
IOH: Input/Output Hub
IPC: Instructions Per Cycle
IPMI: Intelligent Platform Management Interface
IVB: IVy Bridge
L1/L2/L3: Level 1/2/3 cache
L1d / L1i: Level 1 Data cache / Level 1 Instruction cache
LAMMPS: Large-scale Atomic/Molecular Massively Parallel Simulator
LAPACK: Linear Algebra PACKage
LIKWID: Like I Knew What I am Doing
LLC: Last Level Cache / Longest Latency Cache
MIC: Many Integrated Core
MKL: (Intel) Math Kernel Library
MMIO: Memory Mapped I/O

MPI: Message Passing Interface
MSR: Machine/Model-Specific Register
NUMA: Non-Uniform Memory Access
OMP: OpenMP
OpenMP: Open Multi-Processing
OpenMPI: Open Message Passing Interface
OS: Operating System
PAPI: Performance Application Programming Interface
PBS: Portable Batch System
PCH: Platform Controller Hub
PCI: Peripheral Component Interconnect
PCM: Performance Counter Monitor
PCU: Power/Package/Platform Control Unit
PDU: Power Distribution Unit
PECI: Platform Environment Control Interface
PEBS: Precise Event Based Sampling
PLASMA: Parallel Linear Algebra for Scalable Multi-core Architectures
PMC: Performance Monitoring Counter
PMI: Performance Monitoring Interrupt
PMU: Performance Monitoring Unit
PWscf: Plane-Wave Self-Consistent Field
QE: Quantum ESPRESSO
QoS: Quality of Service
QPI: QuickPath Interconnect
RAM: Random-Access Memory
RAPL: Running Average Power Limit
RFTS: Run Fast Then Stop
SDM: (Intel) Software Developer's Manual
SIMD: Single Instruction, Multiple Data
SLURM: Simple Linux Utility for Resource Management
SNMP: Simple Network Management Protocol
(NVidia-)SMI: System Management Interface
SMT: Simultaneous MultiThreading (Hyper-Threading)
SoC: System-on-Chip
SP: Single Precision
SSE: Streaming SIMD Extensions
sysfs: system filesystem
SVID: Serial Voltage ID
TCO: Total Cost of Ownership
TSC: Time Stamp Counter
TDP: Thermal Design Power / Thermal Design Point
TLB: Translation Lookaside Buffer
TMAM: Top-Down Microarchitecture Analysis Method
Uop: Micro-Operation
VM: Virtual Machine
VR: Voltage Regulator

CINECA: Consorzio INteruniversitario per il Calcolo Automatico
CNR-IOM: Consiglio Nazionale delle Ricerche - Istituto per l'Officina dei Materiali
CO.S.IN.T./COSINT: COnsorzio per lo Sviluppo INdustriale di Tolmezzo
DEMOCRITOS: DEMOCRITOS MOdeling Center for Research In aTOmistic Simulation
MHPC: Master In High Performance Computing
Micrel lab: MICroELectronic Laboratory
SISSA: Scuola Internazionale Superiore di Studi Avanzati
UNIBO: UNIversity of BOlogna

Appendix B. TMAM formulas and performance events

The following formulas, extracted from [59], can be used in order to obtain the pipeline slots:

```
SLOTS            = PIPELINE_WIDTH * CPU_CLK_UNHALTED_THREAD
Front-End Bound  = IDQ_UOPS_NOT_DELIVERED_CORE / SLOTS
Retiring         = UOPS_RETIRED_RETIRE_SLOTS / SLOTS
Bad Speculation  = (UOPS_ISSUED_ANY - UOPS_RETIRED_RETIRE_SLOTS +
                   PIPELINE_WIDTH * INT_MISC_RECOVERY_CYCLES) / SLOTS
Back-End Bound   = 1-(Front-End Bound + Retiring + Bad Speculation)
```

`PIPELINE_WIDTH` for Intel Ivy Bridge is 4.

The performance events names can be translated to the following hex flags:

```
--------------------------------------------------
Event Event Event       Event
Code  Mask  Hexcode     name
--------------------------------------------------
3Ch   00h   0x003C      CPU_CLOCK_UNHALTED_THREAD_P
9Ch   01h   0x019C      IDQ_UOPS_NOT_DELIVERED_CORE
C2h   02h   0x02C2      UOPS_RETIRED_RETIRE_SLOTS
0Eh   01h   0x010E      UOPS_ISSUED_ANY
0Dh   03h   0x030D      INT_MISC_RECOVERY_CYCLES
--------------------------------------------------
```

In the TMAM analysis exposed in this work, the aforementioned performance events have been obtained using the hex flags by means of `perf`, and the calculation was demanded to `gnuplot`.

Appendix C. FLOPS from performance counters

Event Code	Event Mask	Event Hexcode	Event name
10h	01h	0x0110	FP_COMP_OPS_EXE.X87 [1]
10h	10h	0x1010	FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE [2]
10h	20h	0x2010	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE [3]
10h	40h	0x4010	FP_COMP_OPS_EXE.SSE_PACKED_SINGLE [4]
10h	80h	0x8010	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE [5]
11h	01h	0x0111	SIMD_FP_256.PACKED_SINGLE [6]
11h	02h	0x0211	SIMD_FP_256.PACKED_DOUBLE [7]

Scalar SP	FP_COMP_OPS_EXE.SSE_FP_SCALAR_SINGLE
Scalar DP	FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE
Packed SP	(FP_COMP_OPS_EXE.SSE_PACKED_SINGLE * 4) + (SIMD_FP_256.PACKED_SINGLE * 8)
Packed DP	(FP_COMP_OPS_EXE.SSE_FP_PACKED_DOUBLE * 2) + (SIMD_FP_256.PACKED_DOUBLE * 4)
Tot SP	Scalar SP + Packed SP
Tot DP	Scalar DP + Packed DP
GFLOPS	(Tot SP + Tot DP) / 1e9 / elapsed_time
GFLOPS #2	(Tot SP + Tot DP + FP_COMP_OPS_EXE.X87) / 1e9 / elapsed_time

By calibrating the obtained results testing known algorithms, it appeared that the obtained count of floating point operations was smaller by a factor 1.2, which was then considered as a multiply factor to obtain the final estimation.
Although the performance results reported for QE and LAMMPS have been obtained and based upon these concepts, further study should be considered to broadly verify this assumption.
Various sources confirm that floating-point performance counters are not reliable, though. [83] [82] [34]

[1] Counts number of X87 uops executed (traditional 8087 style 80bit floating point operations) [negligible impact]
[2] Counts number of SSE* or AVX-128 double precision FP packed uops executed (2x 64bit DP packed into 128 bit register)
[3] Counts number of SSE* or AVX-128 single precision FP scalar uops executed (1x SP operation)
[4] Counts number of SSE* or AVX-128 single precision FP packed uops executed (4x 32bit SP packed into 128 bit register)
[5] Counts number of SSE* or AVX-128 double precision FP scalar uops executed (1x DP operation)
[6] Counts 256-bit packed single-precision floating-point instructions (8x 32bit SP packed into 256 bit register)
[7] Counts 256-bit packed double-precision floating-point instructions (4x 64bit DP packed into 256 bit register)

Appendix D. Unused metrics

This appendix briefly presents the results of the exploratory testing attempted in order to obtain additional power readings concerning the whole system (chassis, blades, network devices) from the external power supply. Reliability and granularity of these readings were assessed (D.1.1). Additional analysis (D.1.2) included a multinode test to validate the power readings obtained by the rectifiers, and a scaling analysis aimed to identify any influence of network-based communication on the overall power consumption.
Moreover, in Appendix D.2., the idle power of the CPUs was measured for each available performance state. In conclusion, the plot reported in Appendix D.3. exposes some of the metrics collected in this project that had to be ignored.

Appendix D.1. Power rectifiers

In a preliminary phase, the monitoring sensors of the 3 power rectifiers available upstream of each chassis were meant to be used in order to detect the power consumption of an entire chassis, the single blades and network devices.
The management interface of the power supply, accessible via SNMP, among several metrics reports the current and the voltage of both the AC input and the DC output, hence the energy absorbed and the power consumption of the system.
Unfortunately, these readings didn't turn out to be accurate enough, and were therefore ignored in the subsequent tests. The following subsection describes and shows the outcome of this test phase.

Even though this test didn't lead to the expected results, the scripts written to query and manage the rectifiers are now part of the management software suite of the infrastructure, now at disposal of the vendor and the system administrators.

Appendix D.1.1. Power consumption detected by the power rectifiers

Some scripts and wrappers were written in order to query the power supply at fixed intervals while performing various sequential operations, in particular the following stages were analyzed:

1. all off;
2. management rootcard powered on;
3. blade #1 on;
4. blade #2 on (#1 on but in idle);
5. ...
6. blade #N on (all previous blades powered on but in idle).

The graph 3.6.1 reports the average power consumption detected on each stage during the power up of the components of the chassis #1, which contains 6 blades (without GPUs).

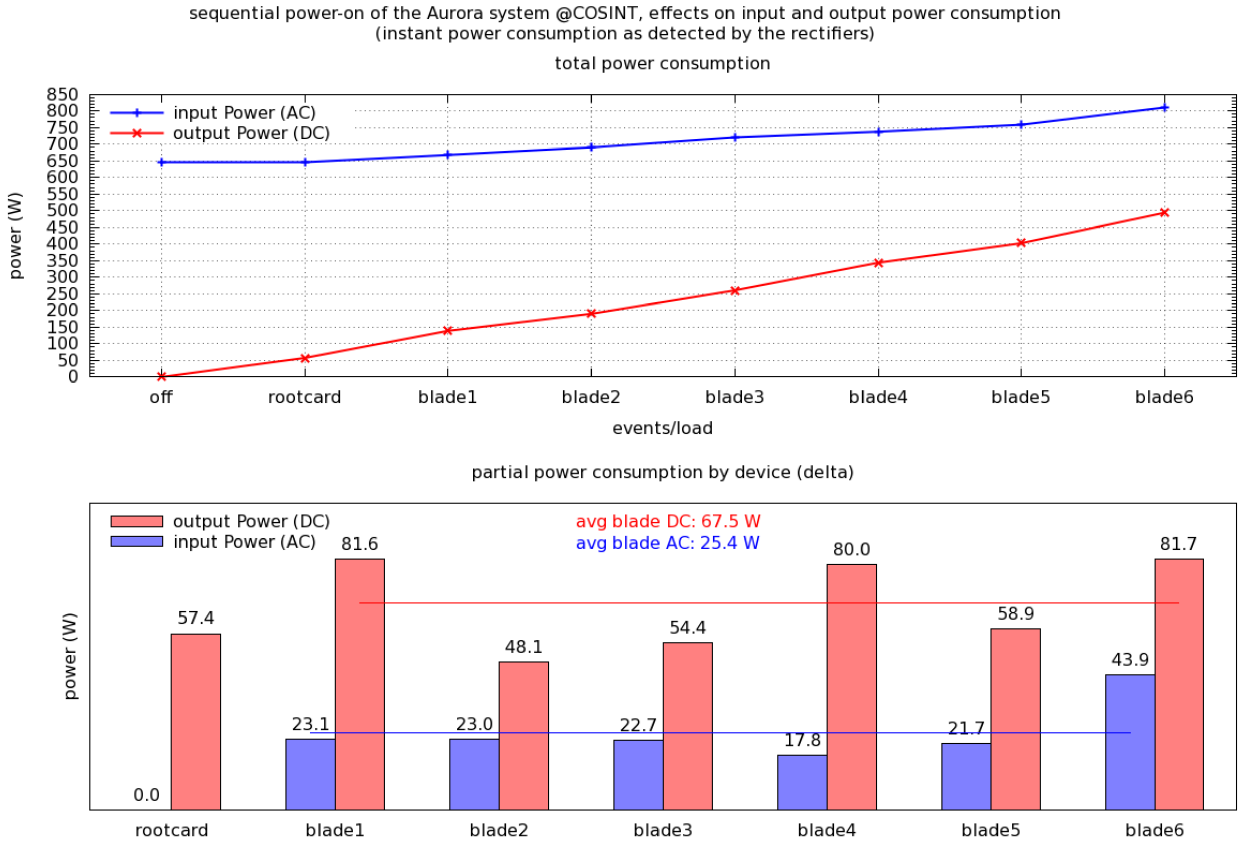


Figure D:1: Power consumption during sequential power-on of chassis #1.

As can be seen from the data obtained, Figure D:1, the power increments almost linearly with the number of blades. Nonetheless, in detail it is possible to observe that the consumption of each blade is different, as reported in the plot at the bottom of D:1, which means that either an average power must be assumed for all the blades or that each one has to be considered on its own.

The rootcard and the chassis components appear to use 50 to 100 W. Each blade powered on, though, does not increase the total for the same amount, as the increments span from 50 to 80 W per blade. Although the boot-up was already completed when each set of data was collected, the reliability of this readings may be of course affected by spurious load of the operating system.

It can also be noted that input and output load of the rectifiers don't appear to be so strictly related, fact that will be more evident in the following test.

For the second chassis, which contains 4 blades with 2 GPUs on each of them, more frequent readings were considered, and various stages of the boot-up reported as well on the graph:

1. hardware power on;
2. IPMI interface booted and accessible;
3. OS still booting, but already replying at `ping` (network configured);
4. OS boot-up completed (`ssh` access and all services up & running, no load beside normal activity).

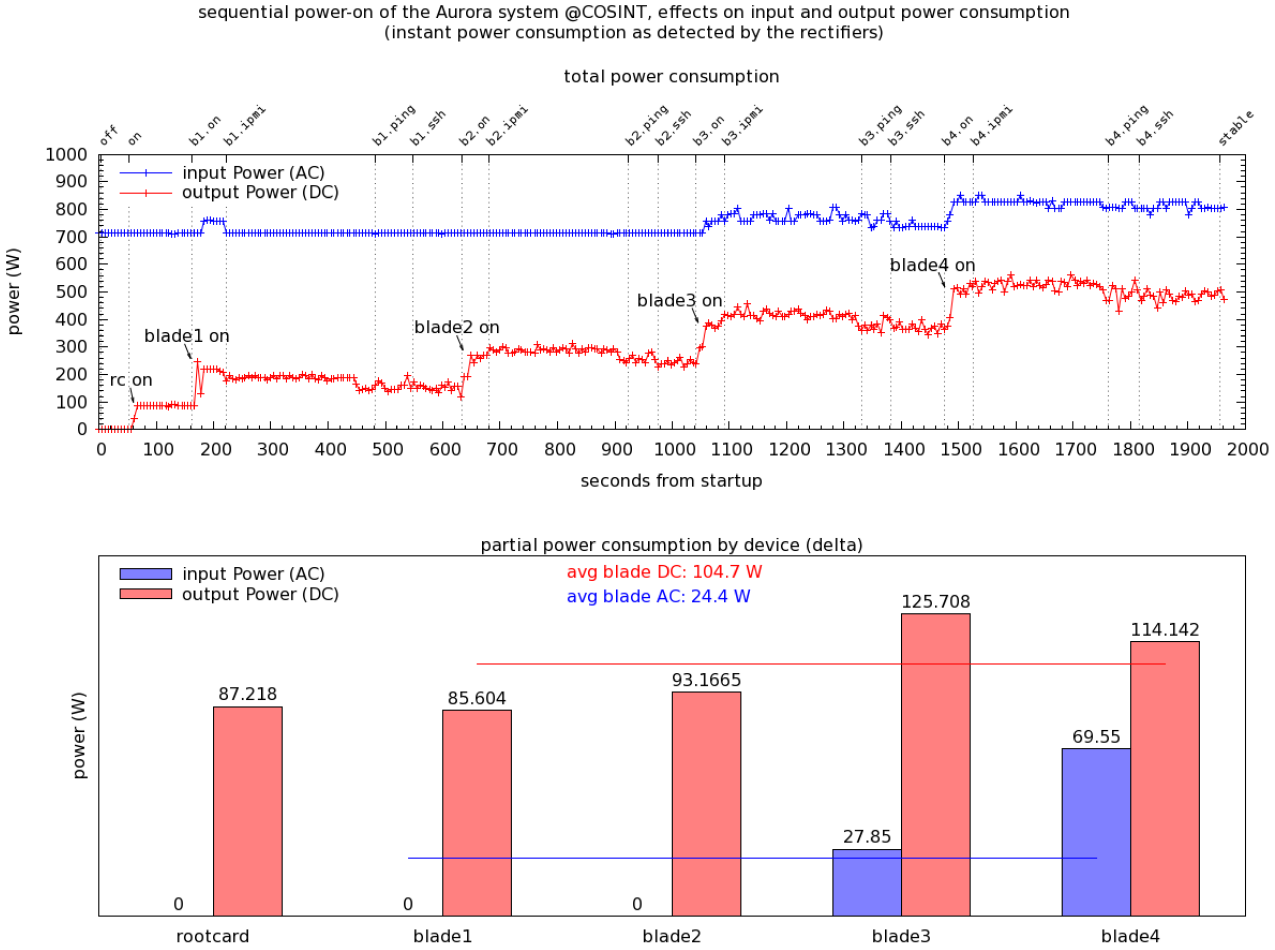


Figure D:2: Power consumption during sequential power-on of chassis #2.

In Figure D:2, although the output power of the rectifiers was increased by the power-on of the rootcard and the first 2 blades, the input power didn't change at all, even though the power consumption of the rootcard and the blades was around 90 W each. An increment can be noticed only when the 3rd blade was switched on.

Should be noted that on the second chassis the blades were not installed on adjacent slots, some slots were left empty and thus the load may not be distributed evenly, hence the surprising result.

Despite the fact that one chassis hosts 6 blades while the second only 4, the overall power consumption is quite close, but the idle power of the 8 GPUs should be taken into account together with the 4 nodes.

Appendix D.1.2. Detecting network devices power consumption using the rectifiers

One of the measures expected to be obtained by using the rectifiers was the power consumption of the network devices (InfiniBand host adapters).
The idea was to detect the power consumption of one or more nodes while running a benchmark (HPL) loading the machine(s) but without any communication, then measure again when running the benchmark in parallel on multiple nodes with communication. The difference was expected to give a rough idea of the consumption demanded by the communication.

This test was performed only on chassis #2 using the 4 blades (#1 had to go back in production), and included the following runs:

- single instance of HPL (24 MPI processes) on a single blade, while the others were in idle state, repeated for each of the 4 blades (24 cores busy overall);
- single instance of HPL (24 MPI processes) on 2 blades at the same time, while the remaining 2 were idling (paired as b21/b22 and b23/b24) (48 cores busy);
- single instance of HPL (24 MPI processes) on 4 blades at the same time (all 96 cores busy);
- single instance of HPL with 48 processes (2 nodes: b21/b22, b23/b24) (48 cores busy);
- single instance of HPL with 96 processes (4 nodes: b21,b22,b23,b24) (all 96 cores busy);
- 2 instances of HPL with 48 processes (2+2 nodes, b21/b22, b23/b24) (all 96 cores busy).

The size of the HPL input was scaled in order to use always the same amount of memory for each MPI process and each node (~54 GB RAM).

The HPL executable used for this test was compiled against MKL. As discussed in section 3.2, MKL delivered the highest performance and the shortest walltime, which was the most important feature required to run such large amount of tests.

In the graph 3.6.2, just for the sake of comparison, each instance of a job spread on multiple nodes is followed by an entry for each single blade involved, computed as the average for the run in question. Furthermore, for the runs with single contemporary instances, the aggregated result is reported too, so that the run with 4 single contemporary instances can be compared to the 2+2 and the 4 parallel instances. Besides giving an idea of the scaling, the difference between this last sets should have led to some insights concerning the power consumption due to the MPI communication over the network.

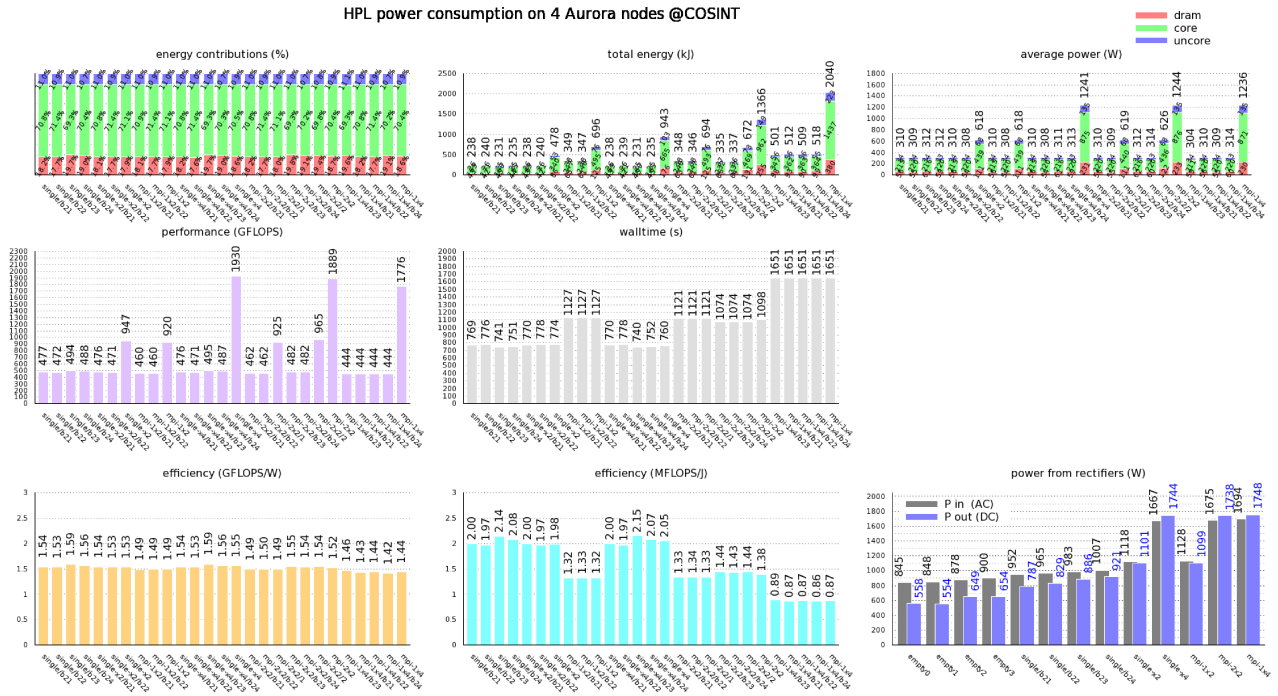


Figure D:3: Power consumption with HPL over multiple nodes.

As expected, the first plot, Figure D:3 top-left, which reports the contributions in percentage for the 3 main CPU sub-systems (CORE, UNCORE, DRAM) shows more or less identical results for all the runs, since the the operations performed are basically the same independently of the number of

MPI processes. The contribution is distributed as 18% from the DRAM, 71% from the CORE and 11% the UNCORE.
The third plot, Figure D:3 top-right, confirms that the CPU is 100% loaded and the computation/utilization pattern is exactly the same for all the instances.
The second plot, Figure D:3 top-center, reports the total energy consumed (kJoules), which is computed as the average power consumption multiplied by the walltime. This plot confirms that what defines the total energy consumption for a CPU-bound application is the time-to-completion.

The forth and fifth plots, Figure D:3 center row, shows the performance in GFLOPS and the walltime as reported by HPL. As expected, the aggregate performance of 4 independent instances is larger than 2 instances of a MPI run spread on 2 nodes, which is in turn larger than a single instance of MPI on all 4 nodes. This difference is obviously due to the communication and the data distribution across nodes, which affect as well the walltime.

Concerning the energy efficiency, expressed as GFLOPS/W in the sixth plot, Figure D:3 bottom-left, there's not much to say. Again, MPI on 4 nodes is less efficient than any other combination due to the overhead of the communication. Plot Figure D:3 bottom-center, express the efficiency as MFLOPS/J, and here, again, multinode MPI jobs result to be less efficient only because of the larger walltime.

Finally, the last plot, Figure D:3 bottom-right, confirms once again that the values extracted from the rectifiers cannot provide any evidence of the power consumption related to the communication, nor give detailed information concerning the single blades. Between single-x4, mpi-2x2 and mpi-1x4, in fact, no significant difference can be highlighted.

The graphs in Figure D:4 report the data acquired from the rectifiers.

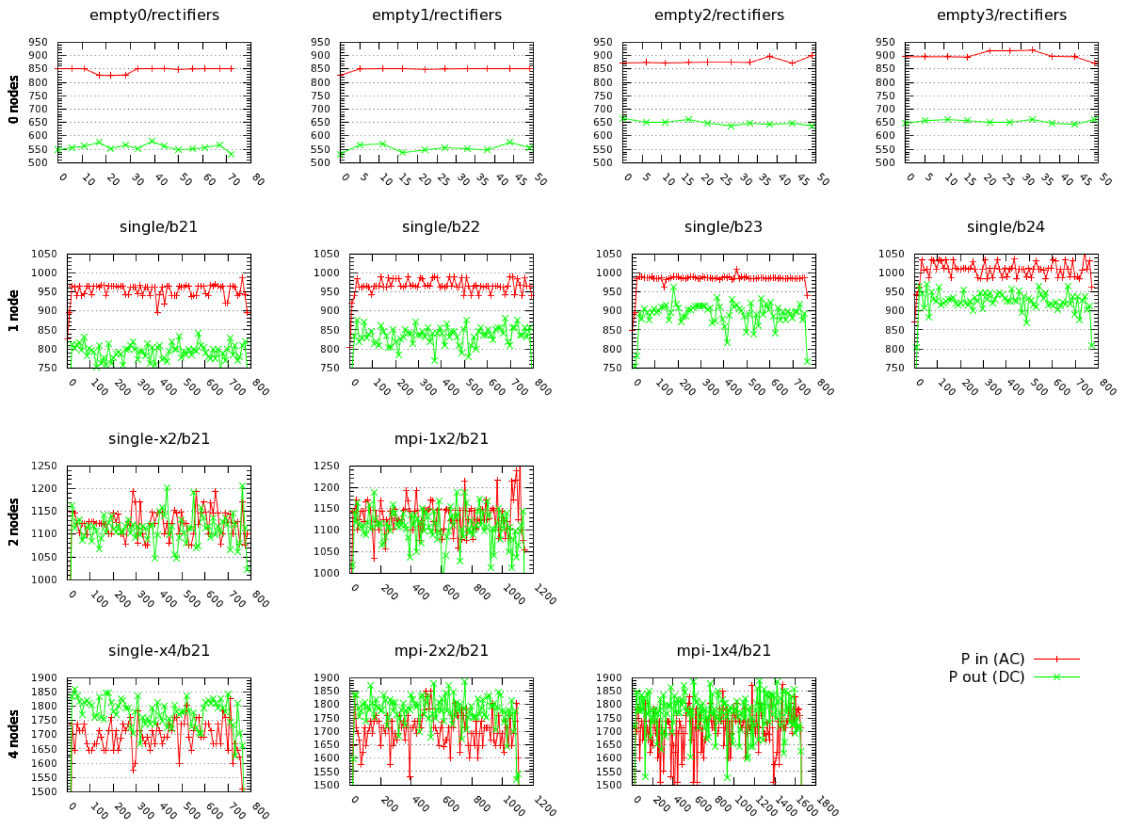


Figure D:4: Power consumption with HPL over multiple nodes as detected by rectifiers.

Appendix D.2. Idle power consumption of the system

This section reports the power consumption of the CPUs detected when the system is idling.

Despite the fact that a specific frequency was imposed through the Linux frequency scaling governor, Figure D:5, shows that the aggressive power saving policies of the CPU uniform the power consumption by putting the unused cores in some sleep state (C-state 1, 3 or 6). The only notable difference is that when Turbo Boost is explicitly chosen as performance state, the power consumption of the DRAM gets slightly higher (+4 W, ~+20%). The `ondemand` governor, 1st bar of both left and right panels, was running the CPU at the minimum frequency (1.2 GHz).

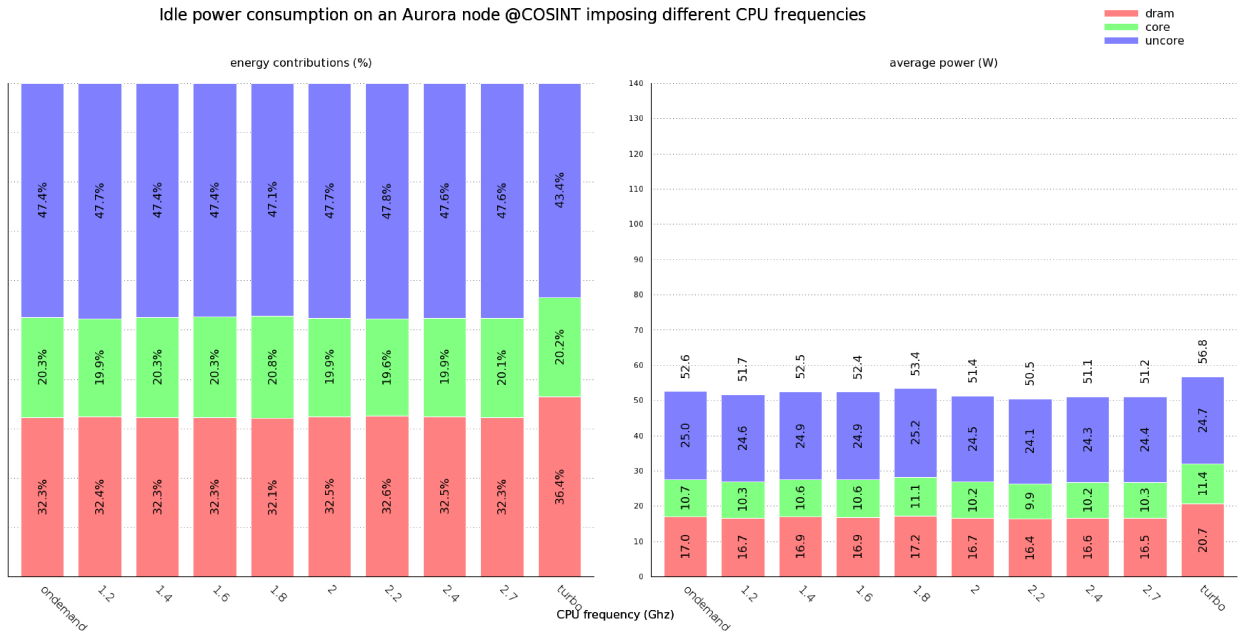


Figure D:5: system overall power consumption when idle (load < 0.3%)

Figure D:6 reports the consumption when a single single-threaded application is demanding the 100% of one core (while the load on the other cores is still <0.3%) . The behavior is closer to what would be expected in case of frequency scaling: the power consumption decreases when the frequency does.
An important thing is highlighted by this test. The `ondemand` governor forces only one core to run at the highest frequency with the Turbo Boost enabled. Nonetheless, the power consumption is lower than what obtained when the Turbo Boost is imposed system-wide. The small delta in power consumption is because in the latter case, also the second socket has the Turbo Boost enabled and therefore presents the same delta observable for DRAM in Figure D:5 in respect of the other frequencies.

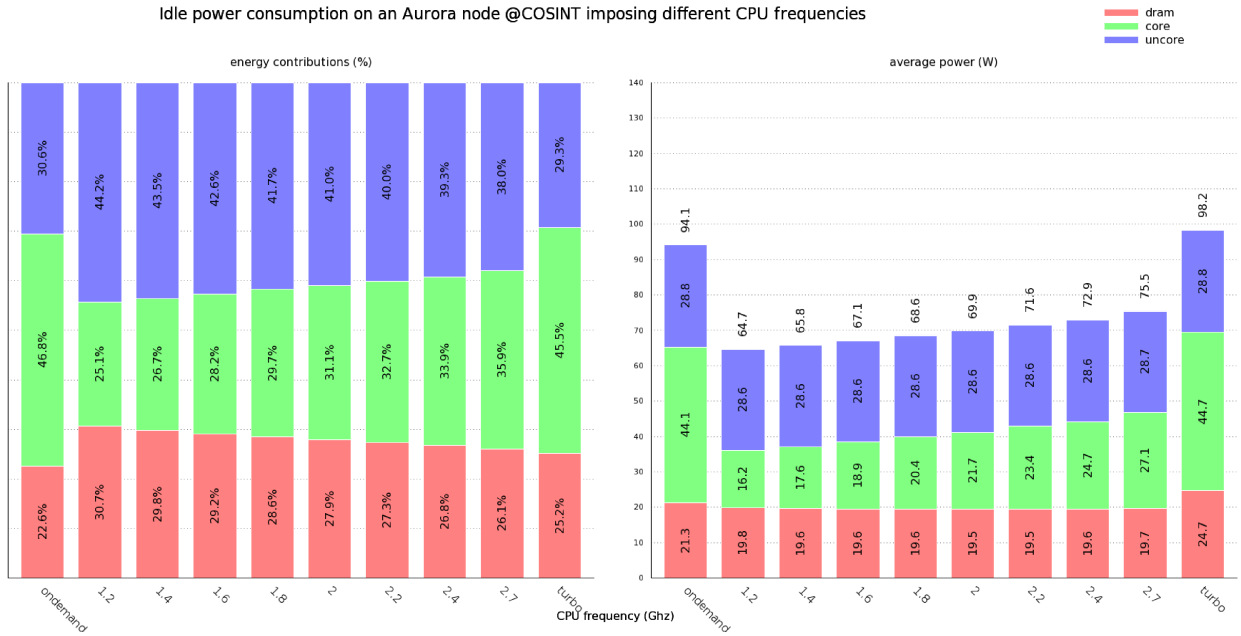


Figure D:6: system overall power consumption with 1 CPU-bound single-threaded application (1 core @99.9%, 23 cores <0.3%)

Figure D:7 confirm this hypothesis. By running two single-threaded applications, each bound to one of the two sockets, `ondemand` highest performance state and `userspace` imposed Turbo Boost show the same power consumption.

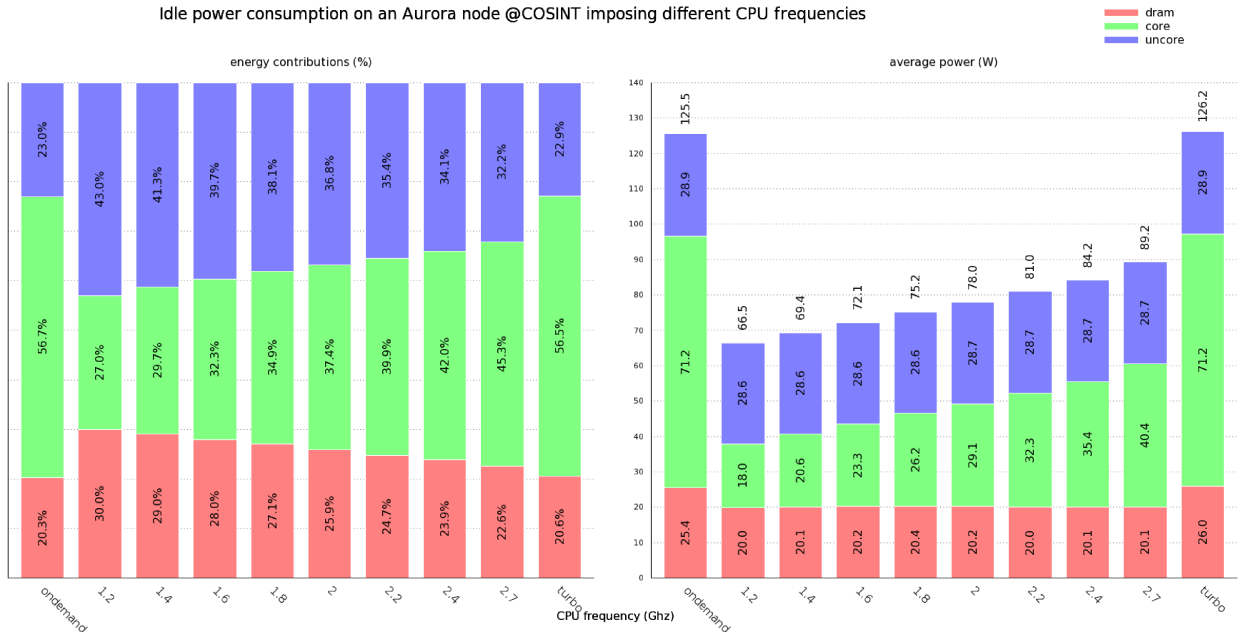


Figure D:7: system overall power consumption with 2 CPU-bound single-threaded application (1 core for each socket @99.9%, 22 cores <0.3%

Appendix D.3. msr-statd (omitted) metrics

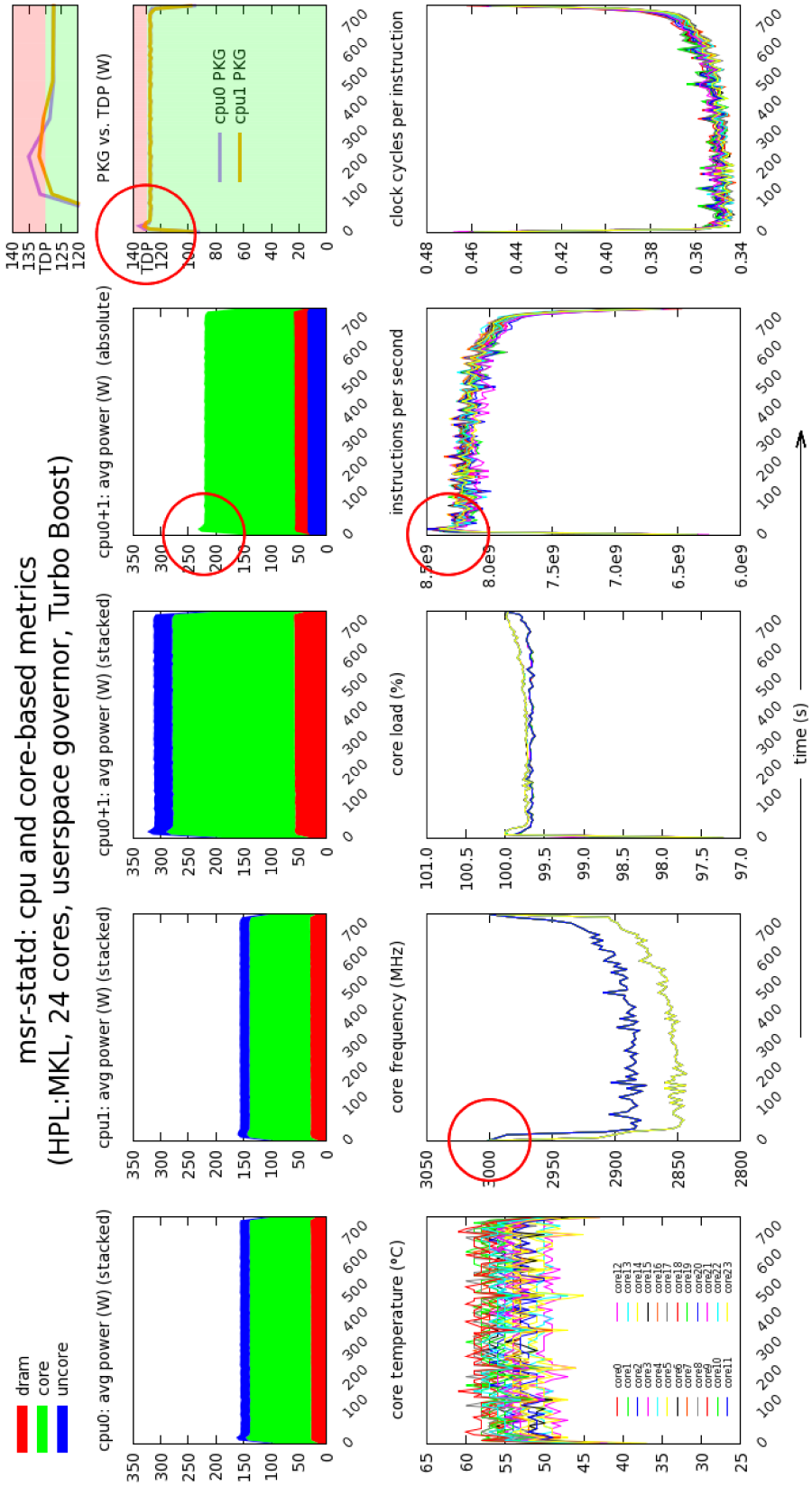


Figure D:8: HPL:MKL, continuous plot of power consumption, frequency scaling, temperature, CPI, IPC, Turbo Boost vs. TDP

Appendix E. Additional plots

Appendix E.1. HPL: power consumption by CPU sub-systems

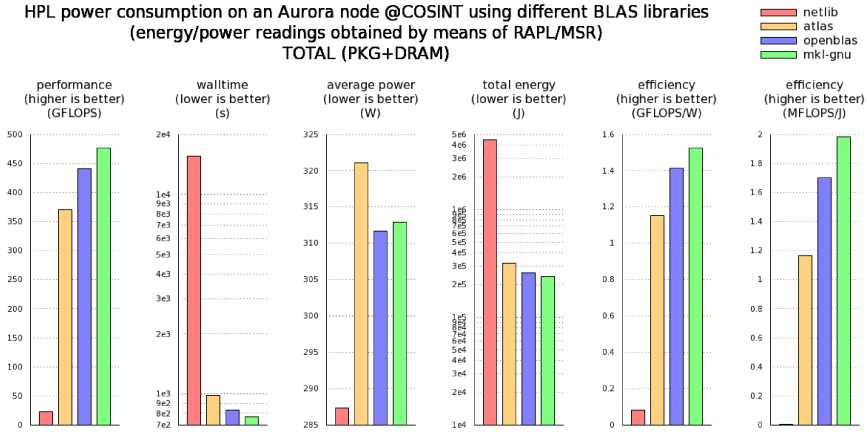


Figure E:1: HPL detail: TOTAL (PKG+DRAM) power consumption

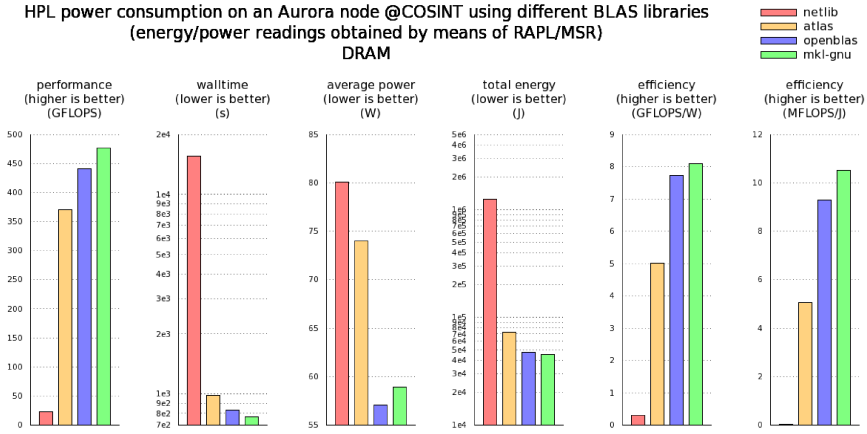


Figure E:2: HPL detail: DRAM power consumption

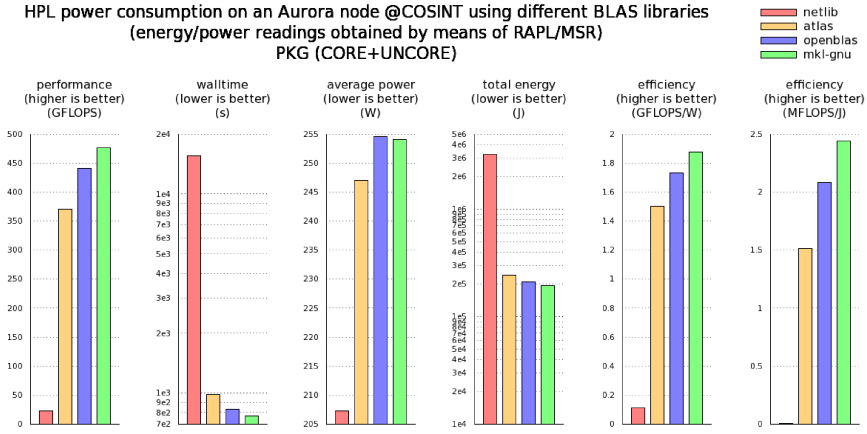


Figure E:3: HPL detail: PKG power consumption

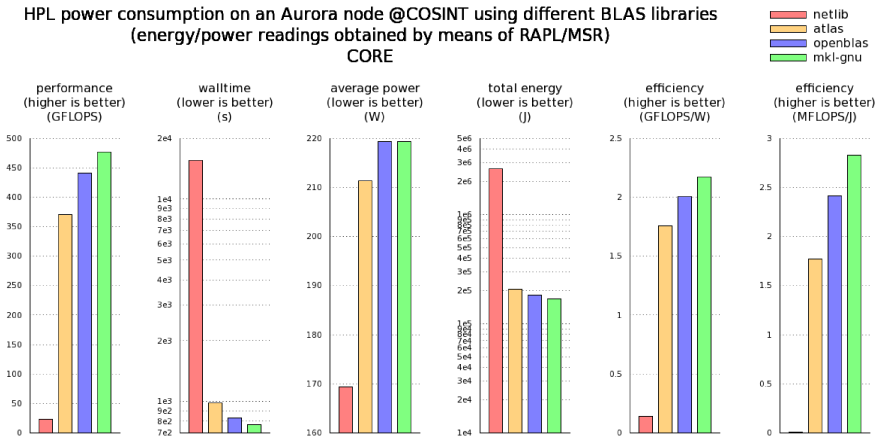


Figure E:4: HPL detail: CORE power consumption

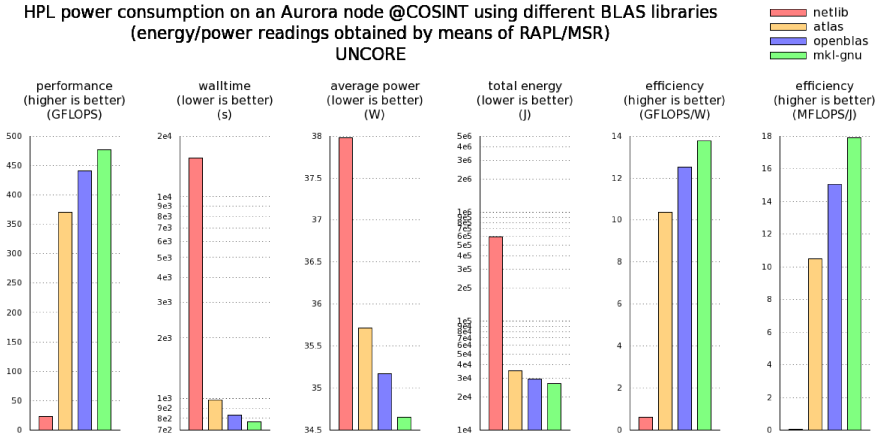


Figure E:5: HPL detail: UNCORE power consumption

Appendix E.2. HPL: problem size scaling

Appendix E.2.1. BLAS comparison

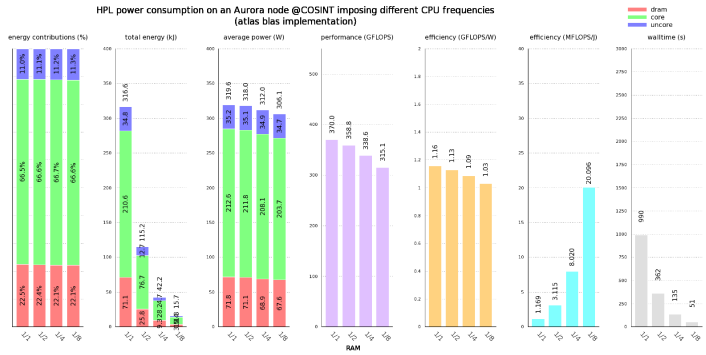


Figure E:6: HPL+ATLAS: problem size scaling

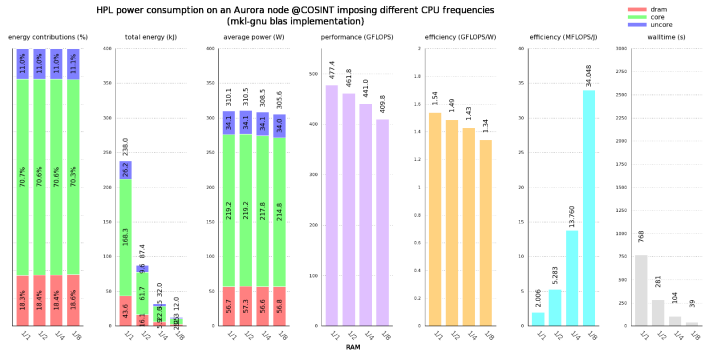


Figure E:7: HPL+MKL: problem size scaling

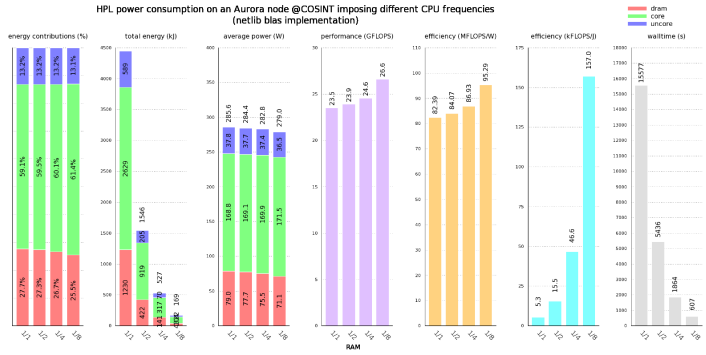


Figure E:8: HPL+NETLIB: problem size scaling

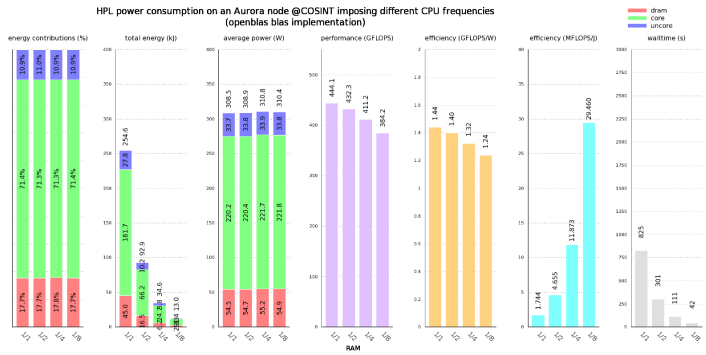


Figure E:9: HPL+OpenBLAS: problem size scaling

Appendix E.2.2. Top-down characterization

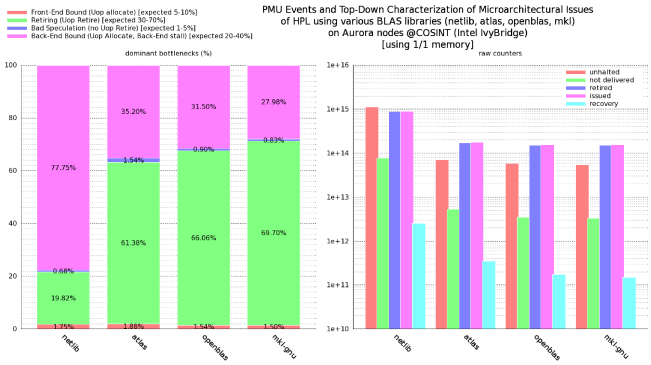


Figure E:10: HPL TMAM: problem size 1/1

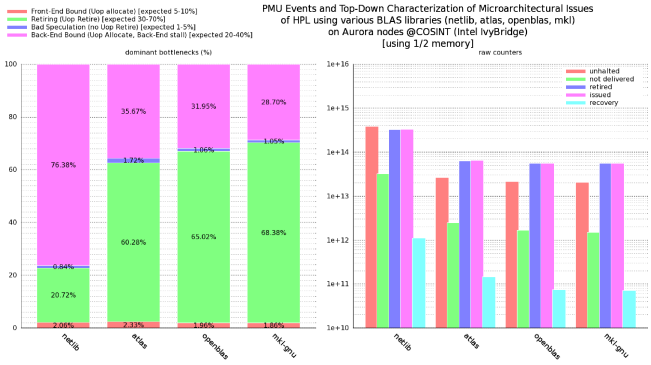


Figure E:11: HPL TMAM: problem size 1/2

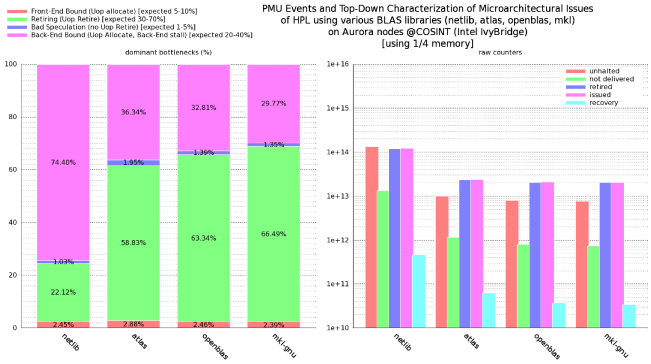


Figure E:12: HPL TMAM: problem size 1/4

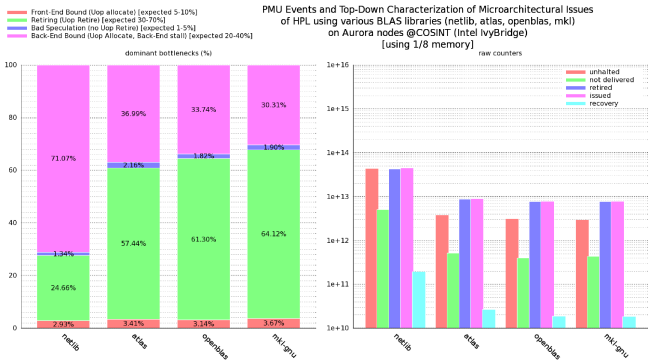


Figure E:13: HPL TMAM: problem size 1/8

Appendix E.2.3. Performance counters

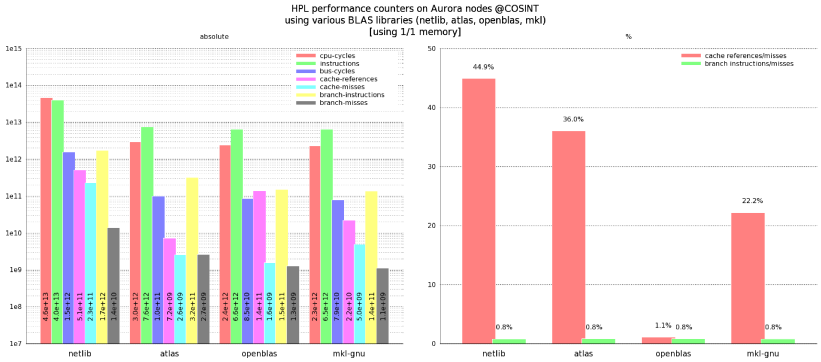


Figure E:14: HPL performance counters: problem size 1/1

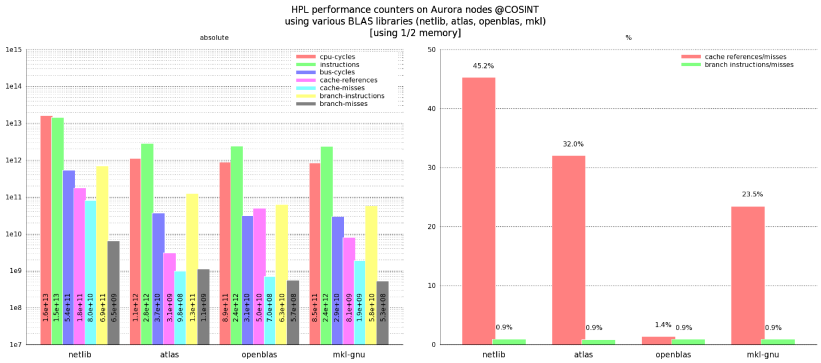


Figure E:15: HPL performance counters: problem size 1/2

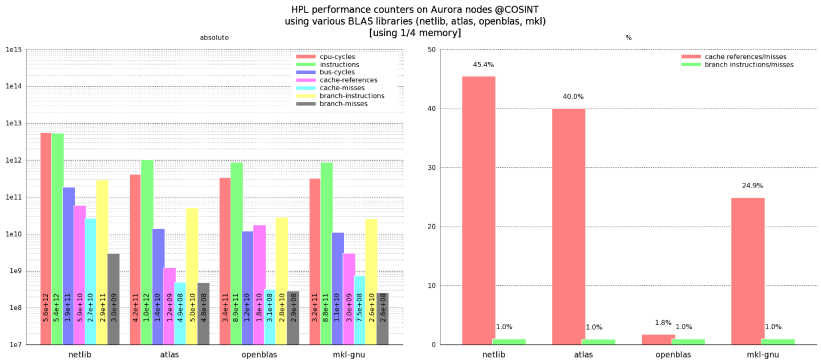


Figure E:16: HPL performance counters: problem size 1/4

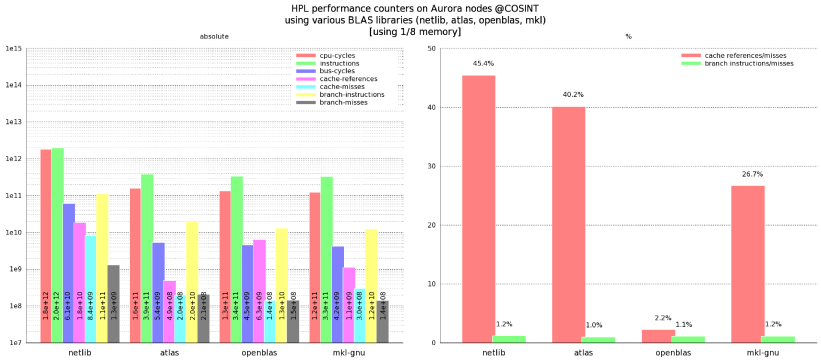


Figure E:17: HPL performance counters: problem size 1/8

Appendix E.3. HPL: problem size and frequency scaling

Appendix E.3.1. ATLAS

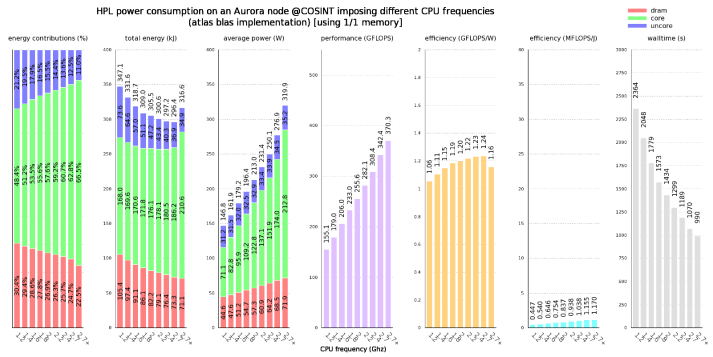


Figure E:18: HPL+ATLAS: size 1/1 vs. frequency

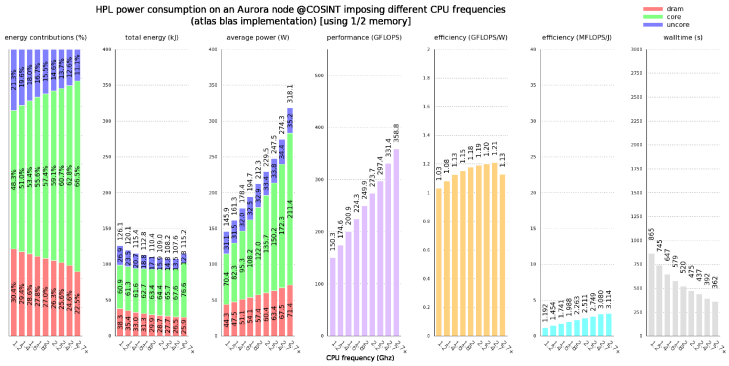


Figure E:19: HPL+ATLAS: size 1/2 vs. frequency

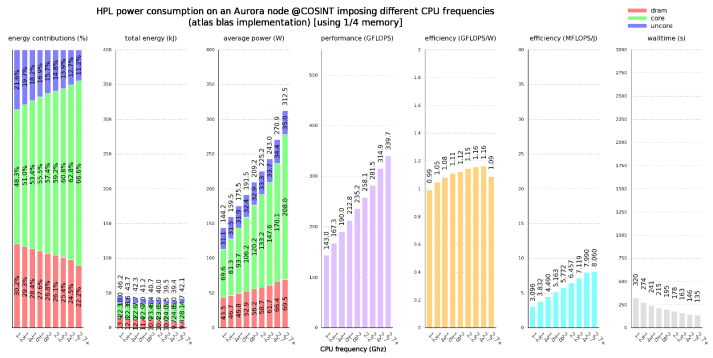


Figure E:20: HPL+ATLAS: size 1/4 vs. frequency

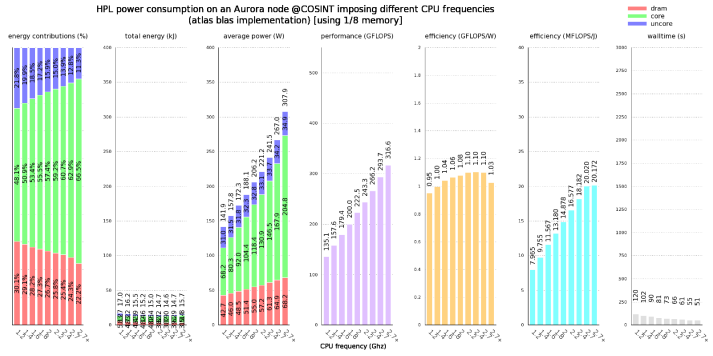


Figure E:21: HPL+ATLAS: size 1/8 vs. frequency

Appendix E.3.2. MKL

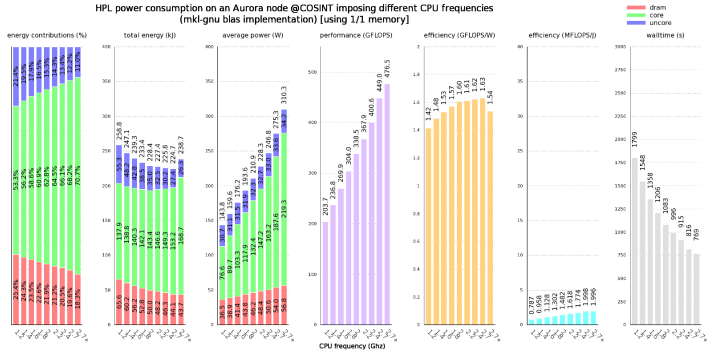


Figure E:22: HPL+MKL: size 1/1 vs. frequency

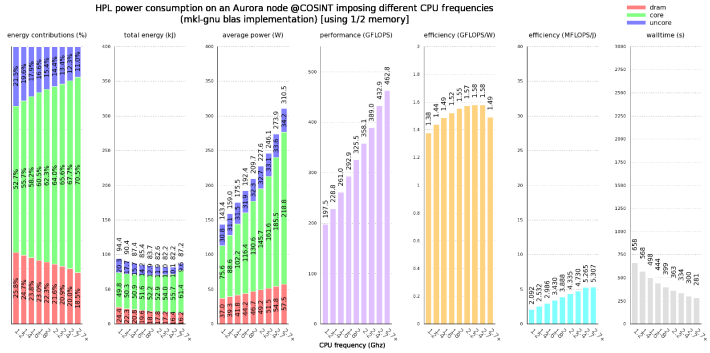


Figure E:23: HPL+MKL: size 1/2 vs. frequency

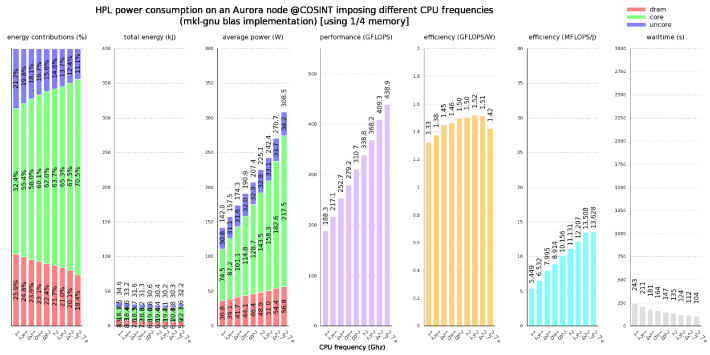


Figure E:24: HPL+MKL: size 1/4 vs. frequency

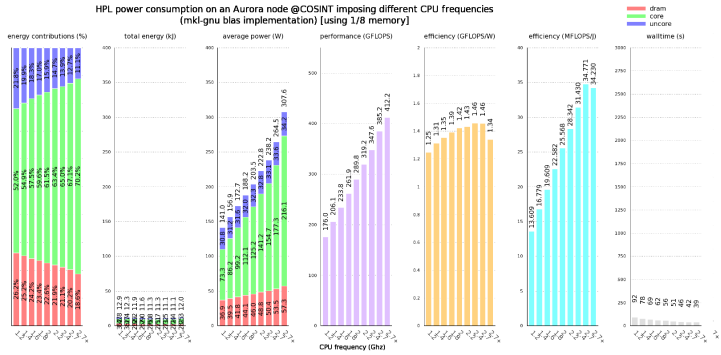


Figure E:25: HPL+MKL: size 1/8 vs. frequency

Appendix E.3.3. OpenBLAS

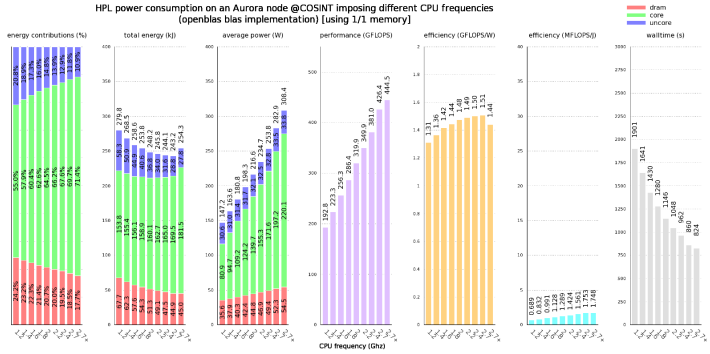


Figure E:26: HPL+OpenBLAS: size 1/1 vs. frequency

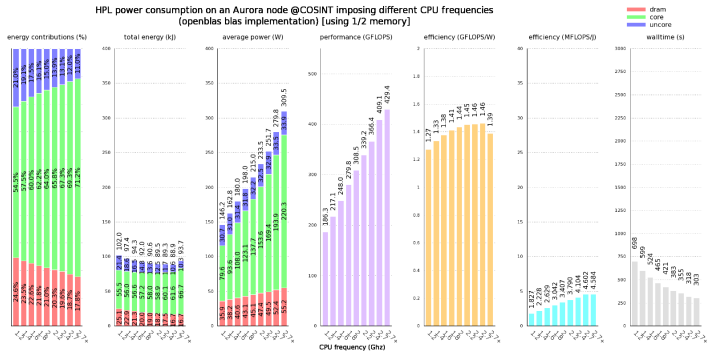


Figure E:27: HPL+OpenBLAS: size 1/2 vs. frequency

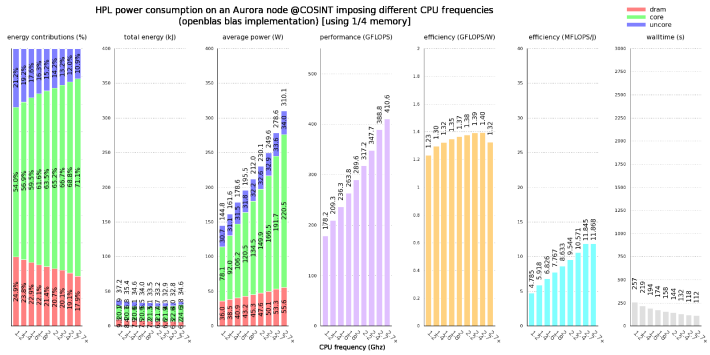


Figure E:28: HPL+OpenBLAS: size 1/4 vs. frequency

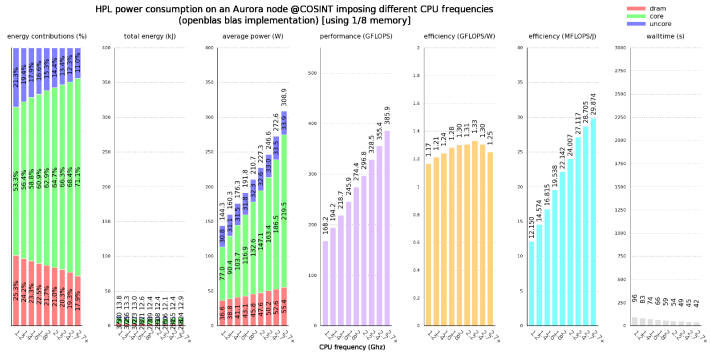


Figure E:29: HPL+OpenBLAS: size 1/8 vs. frequency

Appendix E.3.4. Netlib

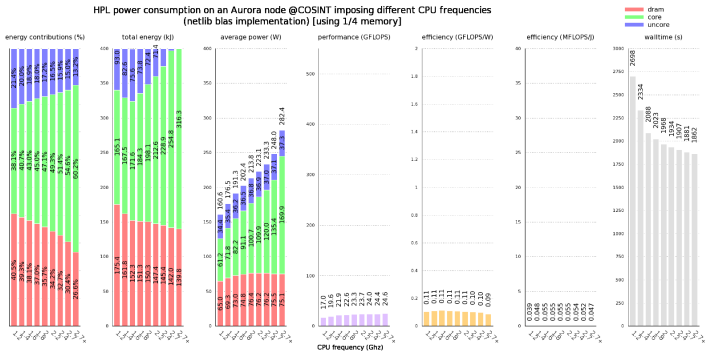


Figure E:30: HPL+Netlib: problem size 1/4 vs. frequency

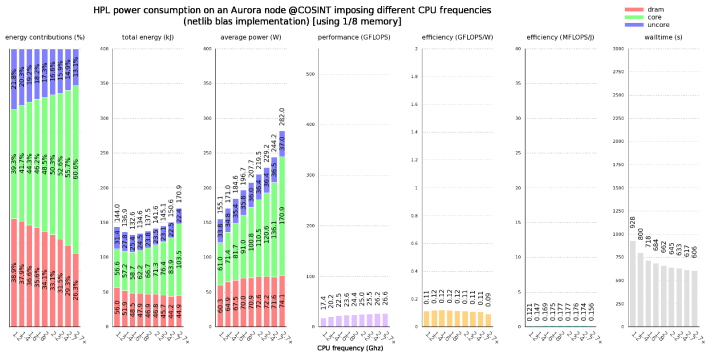


Figure E:31: HPL+Netlib: problem size 1/8 vs. frequency

Note: due to the (extremely) long runtime of HPL+Netlib, no additional tests were performed for 1/1 and 1/2 size.

Appendix E.4. HPCG: problem size and frequency scaling

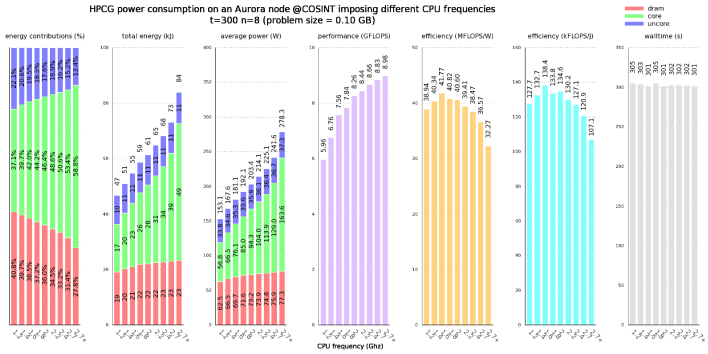


Figure E:32: HPCG: size vs. frequency: n=8

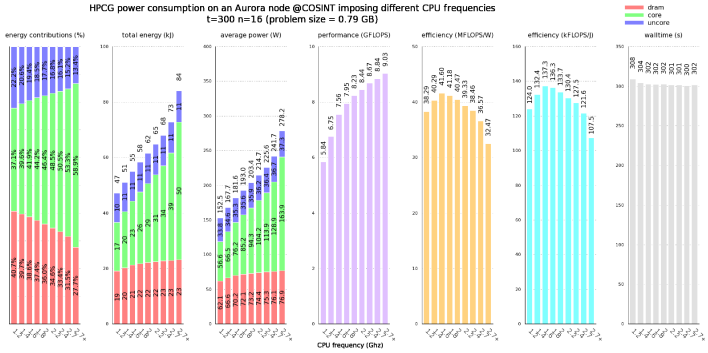


Figure E:33: HPCG: size vs. frequency: n=16

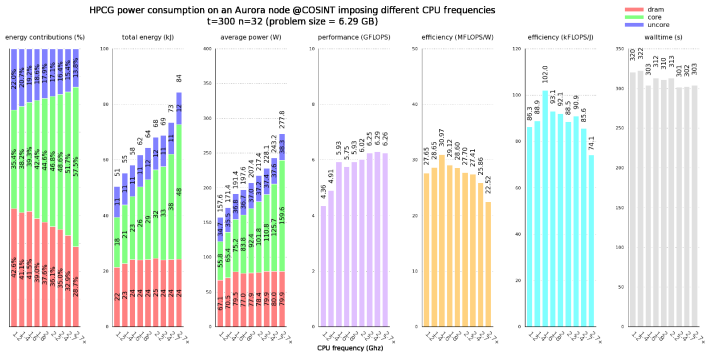


Figure E:34: HPCG: size vs. frequency: n=32

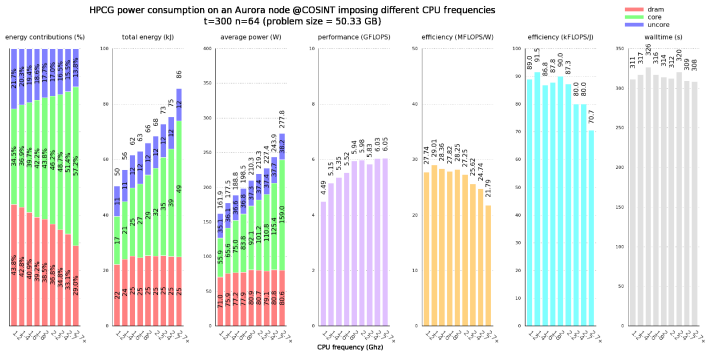


Figure E:35: HPCG: size vs. frequency: n=64

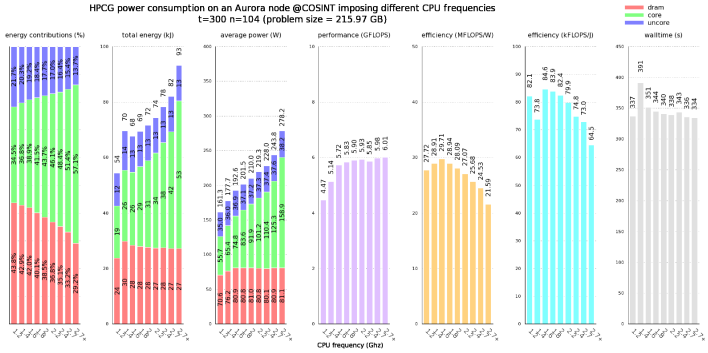


Figure E:36: HPCG: size vs. frequency: n=104

Appendix F. Playing with performance counters

event	mask	hexcode	mnemonic
CBh	01h	0x01CB	MEM_LOAD_RETIRED.L1D_HIT
CBh	02h	0x02CB	MEM_LOAD_RETIRED.L2_HIT
CBh	04h	0x04CB	MEM_LOAD_RETIRED.L3_UNSHARED_HIT
D0h	81h	0x81D0	MEM_UOPS_RETIRED.LOADS
D1h	02h	0x02D1	MEM_LOAD_UOPS.RETIRED_L2_HIT
D1h	04h	0x04D1	MEM_LOAD_UOPS.RETIRED_L3_HIT
D1h	10h	0x10D1	MEM_LOAD_UOPS.RETIRED_L2_MISS
D1h	12h	0x12D1	MEM_LOAD_UOPS.RETIRED_L2_ALL
D1h	20h	0x20D1	MEM_LOAD_UOPS.RETIRED_L3_MISS
D1h	24h	0x24D1	MEM_LOAD_UOPS.RETIRED_L3_ALL
D1h	7Fh	0x7FD1	MEM_LOAD_UOPS.RETIRED_ALL
D2h	01h	0x01D2	MEM_LOAD_UOPS.LLC_HIT_RETIRED_XSNP_MISS
D2h	02h	0x02D2	MEM_LOAD_UOPS.LLC_HIT_RETIRED_XSNP_HIT
D2h	04h	0x04D2	MEM_LOAD_UOPS.LLC_HIT_RETIRED_XSNP_HITM
D2h	08h	0x08D2	MEM_LOAD_UOPS.LLC_HIT_RETIRED_XSNP_NONE
D3h	01h	0x01D3	MEM_LOAD_UOPS.LLC_MISS_RETIRED_LOCAL_DRAM
D3h	0Ch	0x0CD3	MEM_LOAD_UOPS.LLC_MISS_RETIRED_REMOTE_DRAM
D3h	10h	0x10D3	MEM_LOAD_UOPS.LLC_MISS_RETIRED_REMOTE_HITM
D3h	20h	0x20D3	MEM_LOAD_UOPS.LLC_MISS_RETIRED_REMOTE_FWD
F0h	80h	0x80F0	L2_TRANS.ALL_REQUESTS
2Eh	41h	0x412E	L3_LAT_CACHE.MISS (perf "cache-misses")
2Eh	4Fh	0x4F2E	L3_LAT_CACHE.REFERENCE (perf "cache-references")
24h	01h	0x0124	L2_RQSTS.ALL_DEM_AND_DATA_RD_HIT
24h	03h	0x0324	L2_RQSTS.ALL_DEM_AND_DATA_RD
24h	AAh	0xAA24	L2_RQSTS.MISS
40h	01h	0x0140	L1D_CACHE_LD.I_STATE
40h	02h	0x0240	L1D_CACHE_LD.S_STATE
40h	04h	0x0440	L1D_CACHE_LD.E_STATE
40h	08h	0x0840	L1D_CACHE_LD.M_STATE
40h	0Fh	0x0F40	L1D_CACHE_LD.MESI
80h	01h	0x0180	ICACHE.HITS
80h	02h	0x0280	ICACHE.MISSES
80h	03h	0x0380	ICACHE.ACCESSES
80h	04h	0x0480	ICACHE.IFETCH_STALL

```
Various attempts at cache-miss ratio:
L3_ALL     = MEM_LOAD_UOPS.RETIRED_L3_HIT + MEM_LOAD_UOPS.LLC_HIT_RETIRED_XSNP_HIT +
             MEM_LOAD_UOPS.LLC_HIT_RETIRED_XSNP_HITM + MEM_LOAD_UOPS.LLC_MISS_RETIRED_REMOTE_HITM +
             MEM_LOAD_UOPS.RETIRED_L3_MISS
perf L3    = L3_LAT_CACHE.MISS / L3_LAT_CACHE.REFERENCE
L1d        = L1D_CACHE_LD.I_STATE / L1D_CACHE_LD.MESI
L2         = MEM_LOAD_UOPS.RETIRED_L2_MISS / MEM_LOAD_UOPS.RETIRED_L2_ALL
L3         = MEM_LOAD_UOPS.RETIRED_L3_MISS / MEM_LOAD_UOPS.RETIRED_L3_ALL
likwid L1i = ICACHE.MISSES / ICACHE.ACCESSES
likwid L2  = L2_RQSTS.MISS / L2_TRANS.ALL_REQUESTS
forum L2   = 1 - (L2_RQSTS.ALL_DEM_AND_DATA_RD_HIT / L2_RQSTS.ALL_DEM_AND_DATA_RD)
forum L2   = L3_ALL / L2_RQSTS.ALL_DEM_AND_DATA_RD
forum L3   = MEM_LOAD_UOPS.RETIRED_L3_MISS / L3_ALL
```

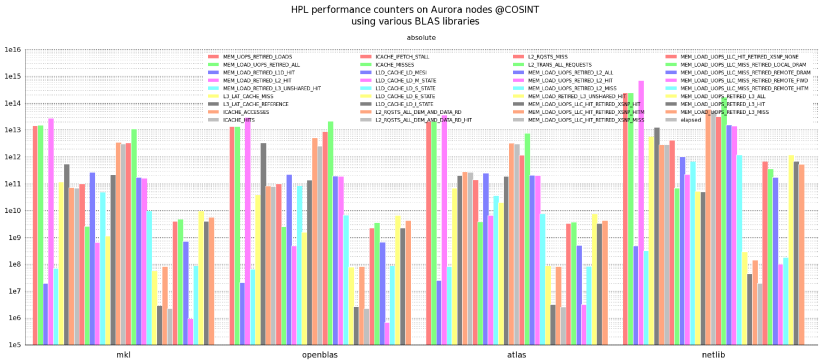


Figure F:1: perfctr: BLAS, longrun#2, L1/L2/L3, raw counters

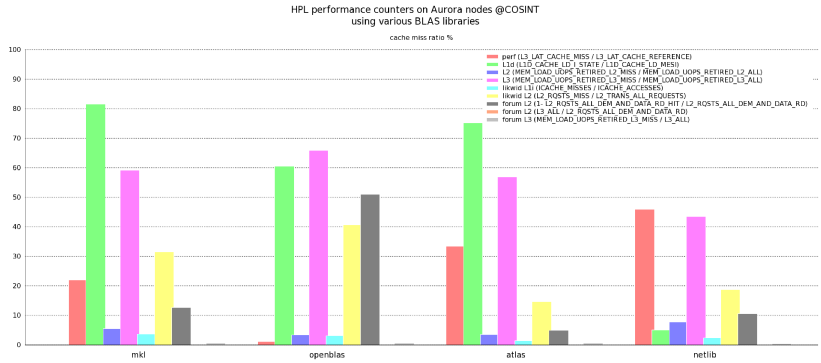


Figure F:2: perfctr: BLAS, longrun#1, L1/L2/L3, ratio

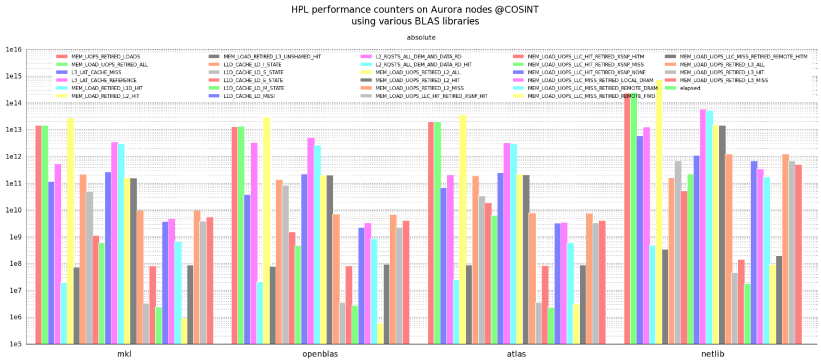


Figure F:3: perfctr: BLAS, longrun#2, L1/L2/L3, raw counters

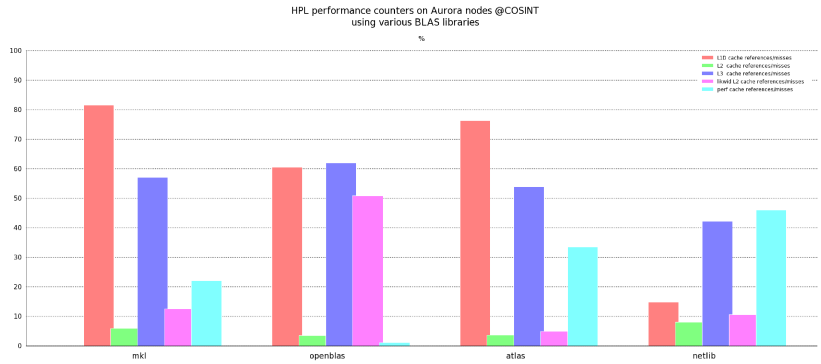


Figure F:4: perfctr: BLAS, longrun#1, L1/L2/L3, ratio

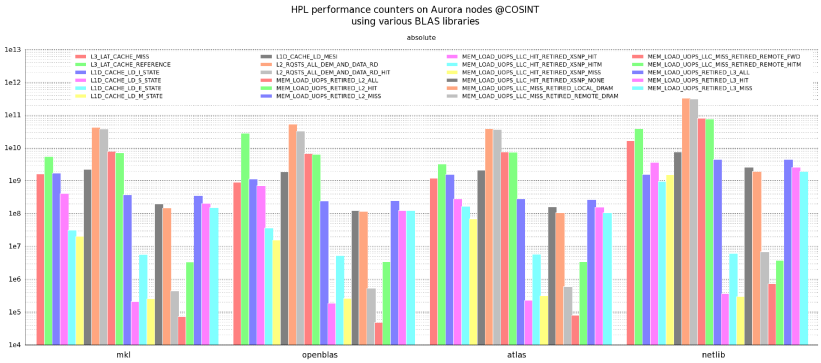


Figure F:5: perfctr: BLAS, shortrun#1, L1/L2/L3, raw counters

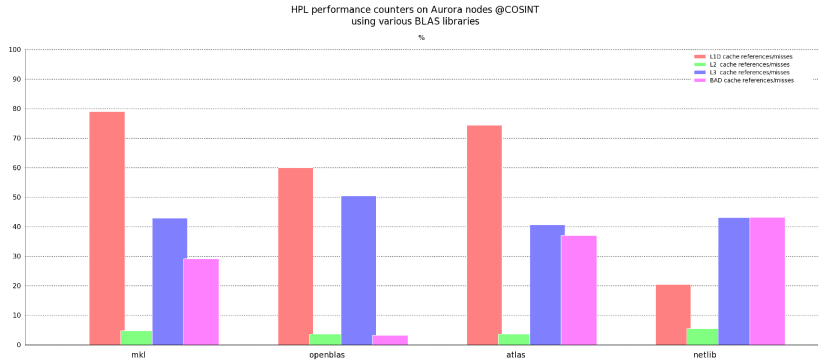


Figure F:6: perfctr: BLAS, shortrun#1, L1/L2/L3, ratio

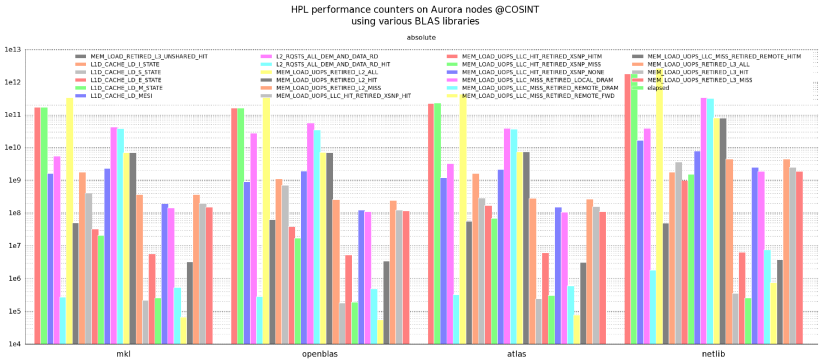


Figure F:7: perfctr: BLAS, shortrun#2, L1/L2/L3, raw counters

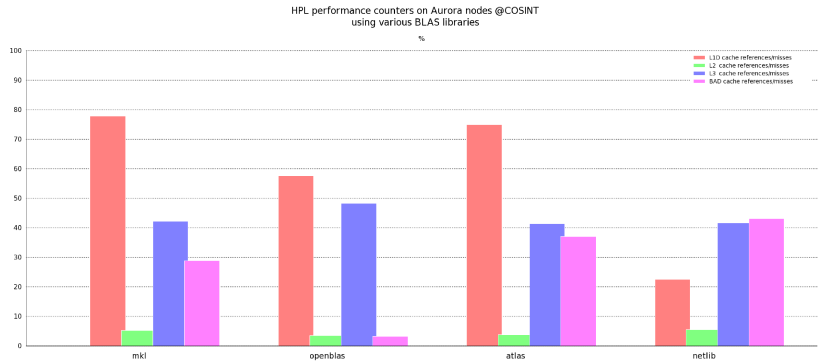


Figure F:8: perfctr: BLAS, shortrun#2, L1/L2/L3, ratio