Green Coding

Software energy optimization by understanding the impact of programming languages

Mohammed Chakib Belgaid

Supervisors: Pr. Romain Rouvoy
Pr. Lionel Seinturier

University of Lille

This dissertation is submitted for the degree of
Doctor of Philosophy

EDSPI June 2022

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Mohammed Chakib Belgaid
June 2022

Acknowledgements

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Contents

List of Figures

List of Tables

Chapter 1

Literature Review

1.1 benchmarking

this is [6]

difference tests [121] [82] [20] list of benchmarks [124]:

- Not evaluating potential performance degradation
- Benchmark subsetting without proper justification
- Selective data set hiding deficiencies
- Microbenchmarks representing overall performance
- Throughput degraded by x% => overhead is %
- Creative overhead accounting
- No indication of significance of data
- Incorrect averaging across benchmark scores
- Benchmarking of simplified simulated system
- Inappropriate and misleading benchmarks
- Same dataset for calibration and validation
- No proper baseline
- Only evaluate against yourself

- Unfair benchmarking of competitors
- Not all contributions evaluated
- Only measure runtime overhead
- False positives/negatives not tested
- Elements of solution not tested incrementally
- Missing platform specification
- Missing software versions
- Subbenchmarks not listed
- Relative numbers only

[101]
startup performance vs steady performance [Buytaert and Eeckhout]
simgrid simulation of power consumption of parallel aplication using simgrid [62]
impact of manifacturing process on the energy variation [32]
different energy consumption between identical processers [idle and hight load][128]
the temperature is the main raison a high energy variation [130]
external factors are impact on the energy variation [99]
the place of the server in the room doesn't affect the energy variation [42]
comparaison of three measuring tools and they did exhebit 10% variation each time [68]
low-cost scalable variation-aware algorithm [68]
reducing the energy variation by disabling turbo boost [3]
reducing the energy consumption in parallel systems [28]
[91]

1.2 energy consumption

to be able to reduce the energy cost of running programs we should first be able to estame this energy consumption. many studies have been conducted to estimate a such energy consumption. that varries from static analysis of the source code to enfere it's energy consumption like Pereira et al. where they provided a tool to highlight the most energy consuming parts of the code [111].the main advantage of this apporach is to be able to estimate the energy consumption of a program without running it. however. unlike the

conplexity of programs, the energy consumption is highly related the execusion environment. Therefor, Static analysis might not represent the real behaviour of the same program when it is run in the production environment. To treat the problem of representativeness, many researchers tend to measure the energy consumption of the programs while they are running them. Hence we will get more accurate results. there is a variety of tools to measure such energy, and they cover a large spectrium of usage depends on how **accurate** and **pricise** those results are needed to be in tone hand, and the price that practinoners are ready to pay for a such accuray/pricision. Below we will present some tools that are well known in the literature and discuss their main advantages / drawbacks for our case.

1.2.1 hardware tools

according to Hackenberg et al. there are four main creterions to evaluate an energy measurement tool [57]

- *spatial granularity: the more specific the target of monitoring we are able to measure, the more efficient we can do optimization since we will know what causes the pitfalls of the energy consumption.*
- temporal granularity: same as spatial granularity, temporal granularity helps us to identif the sequence of code that need to be optimized.
- *scalability: this is mainly related to the cost of the tools and the ease of their integration for our system.*
- accuracy:to elimanate extra hazards and get more representative measurement.

We believe that accuracy can be extracted from those creterions since it is a result of the combaination of the two first ones. therefore, we will focus more on the first 3 creterions from later on. In the follwong section we will discuss some of the well known tools that are used in literature.

Nowdays, most of the high performance computing systems (HPC) implements a tool to report the energy consumption of the nodes for the sake of monitoring and administration. Those tools are mainly integrated withing the power suply units (PSU) or the power distributuin units (PDU). then , they provide an interface and a log to follow the history of the energy consumption. Despite their scalability and ease of integration. such tools lack both in spatial and temporal granularity since they monitor the whole energy of the nodes, and most of the times they have a very low sampling frequency. most of those tools are provided directly by the manufacturers. such as *IBM EnergyScale technology [94] [24] [Caldeira et al.] or* Dell

poweredge [86], MEGware Clusttstafe [18] As we said earlier the true purpose of those tools is more monitoring than analysing the energy consumption. WattsUp Pro, is a device that can be installed between the powerer source of the machine and the system under test. it allows a sampling frequency up to 1Hz and has a internal memory to store a wide variety of data, such as the maximum voltage, current.. etc that later can be exported via USB port for personal usage or lined to some graph programs like Logger Pro or LabQuest.The main advantage of this tool is the ability to monitor the energy consumption from a different device which will reduce the risk of interference with the energy consumption of the test [65]

despite his high temporal granularity , wattsup pro lacks in term of spatial granularity since it monitors the energy consumption of the whole system. To have a finer granularity we need to isolate the energy consumption of each component. . For this we will need more intrusive tools such.

PowerMon and its upgradded version powerMon2 [13] are based on an microcontroler chip that can monitor up to 6 channel (8 for powermon2) simultanetly. therefore we are able to monitor the powerconsumption of 4 devices at the same time. the frequency sample of this tool is up to 50Hz with an accuracy of 1.2%. Powermore2 comes with smaller size that can fit within 3.5 inches rack drive.

PowerInsight [79] is another finegrained measurement tool that is based on an ARM bagelbone processor [33] which is able to measure up to 30 channel simultanetly with a frequency of 1KHz per channel.

on the other hand GreenMiner [64],

powerpack [52] in the other hand is an api that synchronizes the data gathered from other monitoring tools such as Watt's Up Pro, NI and RadioShack pro and the lines of code

other monitor tools have been relized by the manufactures such as IBM Power executive [75] which allows their customer to monitor the power consumption and thermal behaviour of the of BladeCenter systems in the datacenter.

as we see in The CPU is the part responsible of the most energy consumption of softwares. hence the finner we go to measure this energy consumption the better it is for our work. Fortunately, Intel and later AMD proposed a tool that estimate the power consumption of different part of the CPI based on counter performances. RAPL (RUnning Average Power Limit) [56] [58] is a set of registers that was introduced by intel in their CPU since Sandy bridge generation, and later it was fllowed by AMD since Family 17h Zen. It is capable of monitoring different components of CPU , such as DRAM, Cores , and the integrated GPU for desktop processors.

The advanage of a such apporach is that the absence of any intrusive measurement tools. further more they have a high temporal granurality with a sampling frequency up to 1KHz [67].

with similar approach we can find NVIDIA reporting tools such as GPU TESLA [21] and the NVML library [47]

1.2.2 software tools

Now that we got our hands on a precise tools that allows us to monitor

powerapi [34]
smartwatts formula [49]
Wattwatcher [80]
joulemeter [76][71]
jrapl [55] [84]
joulinar [70] [104]
jalen [105]
selfwatts [50]
powerjoular [103]

1.2.3 hybrid tools

1.3 programming languages and performances

1.3.1 energy

energy comparaison of programming languages in the game benchmark by debian [113]

toward green ranking [37]
java vs kotlin in web performances [19]
rosetta code [102] [95]
network energy comparaison [9]
Aequitas [117]
vm placement [97]
Mishra et al.
static cost of Vms [77]
carbon footprint of training neural network [122]
SPELL, the energy leaks detector tool [112]
impact of energy profiling on software [71]

impact of maintability on software energy evolution [25]
defintion [129]
comparaison [27]
emprical study [40]
impact of website implemenation on energy consumption [114] [89]
PHP [15] [39]
simulation of web cache [26]
java spring analysis [51]
performance [96]
cross compiler benchmarking [133]

1.3.2 python

webframeworks on python [Pankiv]
hope library [5]
bytecode effect [14]
numba [38]
intel python [81]
python 2 vs 3 [98]
python runtime performances [116] [100]
static vs dynamic languages [Pang et al.]
concurent benchmark framework [109]

1.3.3 jvm

java vs python [41]
hotspot vs J9 [106] - same but for big querries [30]
constant overheat of the jvm [78]
infer the energy cost based on the byte code instructions [87]
java collection energy footprint [115] [48]
java classes energy footprint [60]
framework to reduce java collections [88]
energy consumption of java dev tools [12]
Microbenchmarks on jvm [85] [12]
android automatic refactoring to reduce energy consumption [11] [118]
java vs native c in android [36]

1.3.4 benchmarking

NAS [Bailey et al.] statistics [61] benchmark crimes [124] microservices [54] impact of webservers on web applications energy [90]

1.4 Reproducibility in benchmarking

1.4.1 Virtual machines

To resolve this problem, practitioners tend to use Virtualisation. Using virtual machines aka VM gives reaserchers the freedom to choose their own tools, software and operating system that they are the most confortable with without paying the price to change the actual working environment, which will give them eventually more controle over the dependencies and the execution environement. Moreover, using a vm will solve the *replication crisis* thanks to the virtual images, even the most complex architecture can be reproduced easily by just instantiating a copy of the image. Since the virtual machines are agnostic to the host architecture, reaserchers won't have to worry about where and how their experiments are replicated because they have already setup the execution environement. Another advantage to the virtual machines is the snapshot mechanism, it allows reaserchers to create backups and revert some changes with simple clicks. Last but not least,thanks to the isolation, virtual machines push the reproducibility further by allowing the future usages to see all the variables -controled and uncontroled- and do other analysis without dealing with any dependencies. In his paper [66] bill howe lists the advantages of using virtual machines in reaserchers experements including the economical impact and cultural limitation to a such approach.

which allow them to have control over the ressouces, the dependencies and the execution environment. Moreover, thanks to the snapshots, deploying a software is easily done by instantiating a copy of that image. However, this choice comes with a certain cost. Because the intervention of the hypervisor,the software will use two kernels , the virtual machine one and the host machine one, which will provide a noticiable overhead, and will impact the performances of the tests.Therefore, we can't use virtual machines for exepements that are related to performance. Another limiation with the virtual machine is the the isolation. It is true that this feature will prevent the experience environement with any undesirable interference from the outside world. but sometimes this contact is needed, especially when the experiment is dependent to an external part, such as sensors. In energicial tests we tend to use hardware powermeters which will make it difficult to use the virtual machines in this case .

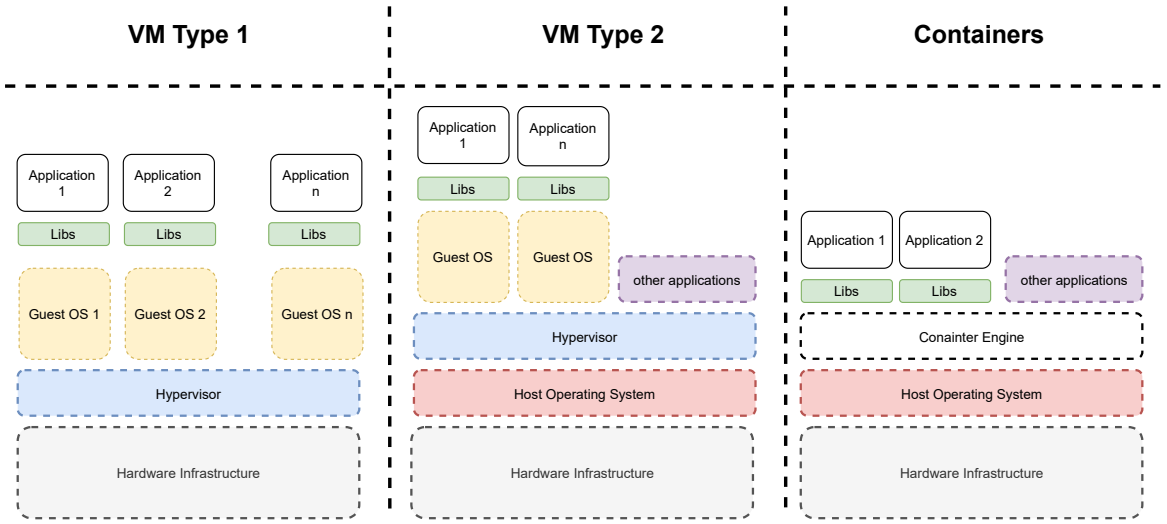

Figure 1.1: Different Methods of Virtualzation

1.4.2 Containers

another solution would be using something that allows us to have the isolation from the host os and the ease of replication that virtual machines offer, and the direct interaction with the hardware that the classical method give. contarization offers a such advantages while keeping the isolation and the ease of replication for application.

Figure 1.1 explains the differents in architecture between the clasic types2 of Virtualisation and Containers.

- Type 1: runs directly on the hardware, it is mainly used by the cloud providers where there is no main OS, but just virtaul machines, we can site for this the open-srouce XEN and VMware ESX

- Type 2: runs over the hostmachine Operating System, mostly used for personal comptuers, VMware server and virtualBox are famous examples of this type, most of the reaserchers expermentation are run witht this type, howerver due to the 2 Operatings syestms the applications tend to be more slower

- containers : Instead of its own kernel, containers used the hots kernel to run their Os, which makes them ligher, quickers and use the full pontial use of the hardware. For this we can cite *Docker*, *Linux LXCLXD* [2]

1.4.3 Docker vs Virtual Machine

Depsite that Type 1 is more performant than type 2, the second one is the most used in reaserch, since most researchers conducts their experements in their own machine. In the other hand, docker is the most famous thechonology for for containers. In ower case we are more prone to docker for two reasons.

1. we need a litetweight orchestrator to not affect the energy consumption of ower tests. As prior work mentioned [cite Morabito (2015) and van Kessel et al. (2016)]

2. since we are using the hardware itself to measure the energy consumption, we are required to interract with the host OS itself.

Special notice to virtualwatts. A framework that allows us to retireve the energy consumption of a virtual machine.

1.4.4 Docker and energy

Now that we have chosen to go with the containers technorolgy to encapsulate our tests. What would be the impact of this solution on the energy consumption of our tests.

Based on the studies of [119], who analysed the impact of adding the docker layer on the energy consumption. In their experement. Eddie Antonio et al run multiple benchmarks with and without docker. and compared their energy consumption and execution time. The first step was to see the impact of docker deamon while there is no work. to see the impact of the orchestrator alone. Later they had the experiment with the following benchmarks

- wordpress
- reddis
- postfresSQL

The following figures represents the energy consumption of the system while it is idle. As we can see in figure 1.2 Docker braught arround 1000joules overhead. In the other hand as we can see in figure 1.3. docker increased the execution time of the benchmark by 50 seconds which caused an increase in energy since they are highly correlated. The authors also highlated the fact that this increase of energy consumption is due to the docker deamon and not the fact that the application is in a container. Moreover they estimted the price of this extra energy and it was less than 0.15$ in the worst case. Which is non significant compared to the advantages that docker bring for isolation and reproducibility.

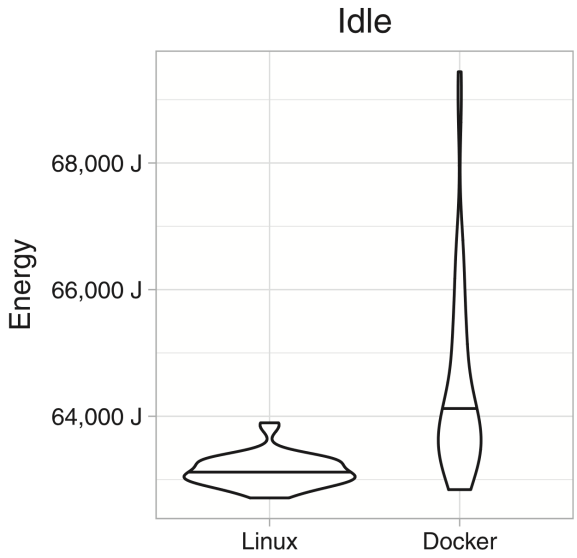

Figure 1.2: energy consumption of Idle system with and without docker [119]

if we recap this study in one sentence,it would be the following one. The dockerised softwares tend to consume more energy, because mainly they take more time to be executed. The average power consumption is higher with only **2Watts** and it is due to the docker deamon.This overhead can be up to 5% for IO intensive application, but it is mearly noticiable when it comes to CPU or DRMA intesive works

1.5 Energy Variation

1.5.1 Studying Hardware Factors

This variation has often been related to the manufacturing process [31], but has also been a subject of many studies, considering several aspects that could impact and vary the energy consumption across executions and on different chips. On the one hand, the correlation between the processor temperature and the energy consumption was one of the most explored paths. Kistowski *et al.* showed in [72] that identical processors can exhibit significant energy consumption variation with no close correlation with the processor temperature and performance. On the other hand, the authors of [131] claimed that the processor thermal effect is one of the most contributing factors to the energy variation, and the CPU temperature and the energy consumption variation are tightly coupled.

TODO : add the corelation value

This exposes the processor temperature as a delicate factor to consider while comparing energy consumption variations across a set of homogeneous processors.

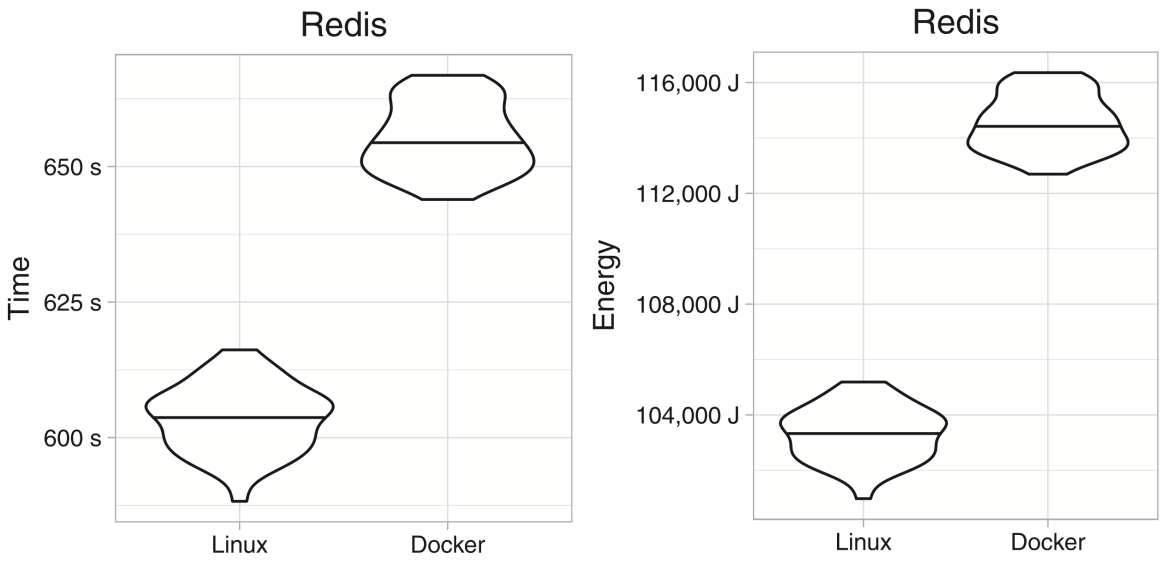

Figure 1.3: execution time and energy consumption of Redis with and without docker [119]

The ambient temperature was also discussed in many papers as a candidate factor for the energy variation of a processor. In [126], the authors claimed that energy consumption may vary due to fluctuations caused by the external environment. These fluctuations may alter the processor temperature and its energy consumption. However, the temperature inside a data center does not show major variations from one node to another. In [46], El Mehdi Dirouri *et al.* showed that switching the spot of two servers does not affect their energy consumption. Moreover, changing hardware components, such as the hard drive, the memory or even the power supply, does not affect the energy variation of a node, making it mainly related to the processor. This result was recently assessed by [131], where the rack placement and power supply introduced a maximum of 2.8 % variation in the observed energy consumption.

Beyond hardware components, the accuracy of power meters has also been questioned. Inadomi *et al.* [69] used three different power measurement tools: RAPL, Power Insight[1] and BGQ EMON. All of the three tools recorded the same 10 % of energy variation, that was supposedly related to the manufacturing process.

1.5.2 Mitigating Energy Variations.

Acknowledging the energy variation problem on processors, some papers proposed contributions to reduce and mitigate this variation. In [69], the authors introduced a variation-aware algorithm that improves application performance under a power constraint by determining

[1]https://www.itssolution.com/products/trellis-power-insight-application

module-level (individual processor and associated DRAM) power allocation, with up to 5.4× speedup. The authors of [59] proposed parallel algorithms that tolerate the variability and the non-uniformity by decoupling per process communication over the available CPU. Acun *et al.* [4] found out a way to reduce the energy variation on Ivy Bridge and Sandy Bridge processors, by disabling the Turbo Boost feature to stabilize the execution time over a set of processors. They also proposed some guidelines to reduce this variation by replacing the old—slower—chips, by load balancing the workload on the CPU cores and leaving one core idle. They claimed that the variation between the processor cores is insignificant. In [29], the researchers showed how a parallel system can be used to deal with the energy variation by compensating the uneven effects of power capping.

In [92], the authors highlight the increase of energy variation across the latest Intel micro-architectures by a factor of 4 from Sandy Bridge to Broadwell, a 15 % of run-to-run variation within the same processor and the increase of the inter-cores variation from 2.5 % to 5 % due to hardware-enforced constraints, concluding with some recommendations for Broadwell usage, such as running one hyper-thread per core.

1.6 Conclusion

As we have seen in the state of the art, many methodes have been proposed to reduce the energy footprint of the the ICT,which they can be applied in deffirent parts of the lifecycle of the program, from consumption to the execution. Furthermore, the execution phase took attention of many researchers because it's the part where the most energy is consumed. This thesis will focus on that aspect as well. However, unlike the most work that have been done on the hardware aspect, we will target the software impact on this energy, starting from the choice of the programming language up to how to tune some features of a frameworks in order to make the software consumes less energy.To do so we use the emprical apporach due to it's concinstency for the moment.Unlike the perfromance which essentialy related to the complexity of the algorithm, the energy consumption is more impacted by the hardware. Therefore, to optimize the enrgy consumption we choose a spiral methode. based on 3 phases:

1. execute the code
2. measure the program
3. infere the guidelines

the aim of this work is to present a set of guidelines to create a benchmarking system to to measure the energy consumption of different programs. After that, we will use this system

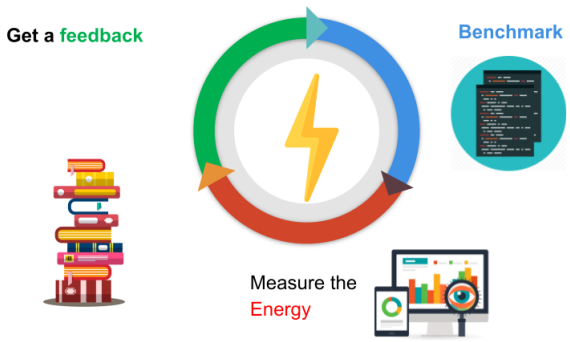

Figure 1.4: the spiral methode of energy optimization

to compare the energy consumption of different programming languages. We will extend the work of Pereira et al. to a closer distance to production environment by comparing a set of usecases. starting by GRPC framework, and a set of Web Framewroks. Finally we will discuss the impact of the execution environment on the energy consumption of two of the most famous programming languages JAVA and Python. and present how can tunning the Virtual Machine can reduce the energy consumption.

Chapter 2

Benchmarking Protocol for measuring Energy Consumption of stoftwares

2.1 Introduction

To optimize the software energy consumption, a reproducible setup environment requires to be designed.

This chapter will be dedicated to provide a set of guidelines to provide a *reproducible, accurate and* representative tests. Each aspect will be detailed in the sections below.

2.2 Threats and Challenges

A successful benchmark has three criterions to fullfil. First, it has to be *reproducible* for others to replicate. Second, the results should be *accurate*, which means each time we rerun the benchmark we are expecting the same results. Finally, it should *represent* the real world. In other words, the conclusions brought from the experiment should be valid outside the experimentation as well. In my case, the real world is the production environment. Therefore, our experiments should reflect what is happening in the production environments. In this section, I will deeper dig in each aspect, present what has been done by others and my contributions, and finally I will propose some perspectives, to improve such experiments.

2.2.1 Reproducibility

One of the most difficult challenges faced by researchers is the reproducibility of their benchmarks. Actually, many results fail to be reproduced,[1] which led to a *replication crisis*. As this crisis hit most of the empirical studies, most of the reviews now includes reproducibility as one of the minimal standard for judging scientific merit.[110] One of the criterions supporting reproducibility is the publication of the dataset and the algorithms run on the raw data to conclude the results. There is even some disagreement about what the terms "reproducibility" or "replicability" by themselves mean [53]. According to [43], *replicability* extends *reproducibility* with the ability to collect a new raw dataset comparable to the original one by re-executing the experiment under similar condition, instead of just the ability to get the same results by running the statistical analyses on the original data set.

In the area covered by this PhD thesis, reproducibility might be achieved by ensuring the same execution settings of physical nodes, virtual machines, clusters or cloud environments. However, when it comes to measuring the energy consumption of a system, applying acknowledged guidelines and carefully repeating the same benchmark can nonetheless lead to different energy footprints not only among homogeneous nodes, but even within a single node.

One major problem that hinders the reproducibility of the empirical benchmarks is the interaction with the external environment, either as concurrency or dependencies. Therefore, researchers cannot observe the same results, unless they duplicate the same environment.

2.2.2 Accuracy

According to Oxford, *accuracy* means "technical The degree to which the result of a measurement, calculation, or specification conforms to the correct value or a standard". In ny case, this means the ability to run the benchmark multiple times with a low variation.

Recently, the research community has been investigating typical "crimes" in systems benchmarking and established guidelines for conducting robust and reproducible evaluations [125].

In theory, using identical CPU, same memory configuration, similar storage and networking capabilities, should increase the accuracy of physical measurements. Unfortunately, this is not possible when it comes to measuring the energy consumption of a system. Applying the benchmarking guidelines and repeating the same experiment with in the same configuration are not sufficient to reproduce the the same energy measurements, not only between identical

[1] Trouble at the lab, The Economist, 19 October 2013; www.economist.com/news/briefing/ 21588057-scientists-think-science-self-correcting- alarming-degree-it-not-trouble.

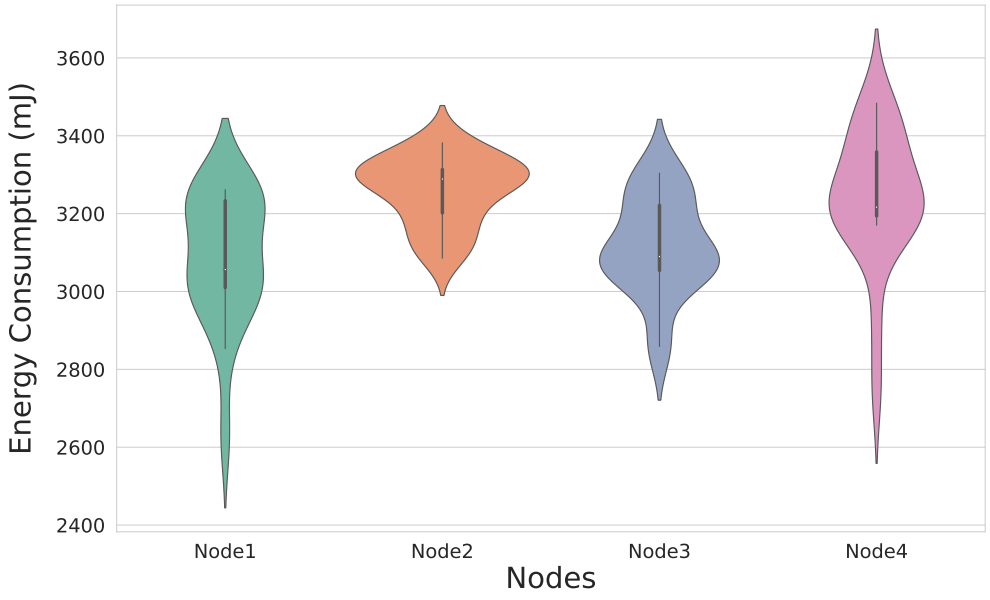

Figure 2.1: CPU energy variation for the benchmark CG

machines, but even within the same machine. This difference—also called *energy variation* (EV)—has become a serious threat to the accuracy of experimental evaluations.

Figure 2.1 illustrates this variation problem as a violin plot of 20 executions of the benchmark *Conjugate Gradient* (CG) taken from the *NAS Parallel Benchmarks* (NBP) suite [8], on 4 nodes of an homogeneous cluster (the cluster Dahu described in Table 2.1) at 50 % workload. We can observe a large variation of the energy consumption, not only among homogeneous nodes, but also at the scale of a single node, reaching up to 25 % in this example.

Some researchers started investigating the hardware impact of the energy variation of power consumption. As an example, one can cite [17, 123] who reported that the main cause of the variation of the power consumption between different machines is due to the **CMOS** manufacturing process of transistors in a chip. [63] described this variation as a set of parameters, such as CPU Frequency and the thermal effect.

2.2.3 Representativeness

As obvious as it seems, the reason of executing benchmarks is to validate ideas so we can use them in the real life. However, this is means that those benchmarks have to represent reality somehow. Basically, when we aim to benchmark something, we create a mock up

version of the situation that we want it to work. First, we can talk about the benchmarking and their selection, then we will talk about the stress benchmark for some applications, and finally we will bring this representativeness in our case and how can we get closer to the energy consumption behavior of a software between the benchmark environnement and the production one.

As Stephen M. Blackburn *et al.* cited in their paper "evaluate collaboratory" [16], one of the major pitfalls of the measurement contexts is the inconsistency, which can be translated here by the fact that the production context is not the same as the benchmarking one.

Another difficult part for practitioners is to generalize the claims they reached beyond the lab conditions. Are they appropriate? Are they consistent and are they reproducible? To answer those questions, the community agreed on some wellknown benchmarks to represent a specific concern of the production world. One can site as an example the Dacapo and Renaissance benchmark suites for Java applications, or the CLBG benchmark suite for comparing programming languages. Although they do not cover all the cases, the community agrees on their relevance and representativeness.

In addition to those benchmarks, a new category of benchmarking technique emerged to simulate the worst case of the production environments, inlcuding performance tests, which are benchmarks meant to evaluate the behavior of a software under stress situations. We can cite as an example the Gatling for web application and stress-ng for hardwares.

2.2.4 docker and accuracy

And now Since the stat of the art has aggreed on the impact of docker on the energy consumption,Let's discuss it's impact on the accuracy. In other words
RQ : does Docker affect the energy variation of the exepements ?

To Answer this question we have conducted a preliminary experiment by running the same benchmarks LU, CG and EP in a Docker container and a flat binary format on 3 nodes of the cluster Dahu to assess if Docker induces an additional variation. Figure 2.2 reports that this is not the case, as the energy consumption variation does not get noticeably affected by Docker while running a same compiled version of the benchmarks at 5 %, 50 % and 100 % workloads. In fact, while Docker increases the energy consumption due to the extra layer it implements [44], it does not noticeably affect the energy variation. The *standard deviation* (STD) is even slightly smaller ($STD_{Docker} = 192mJ$,$STD_{Binary} = 207mJ$), taking into account the measurements errors and the OS activity.

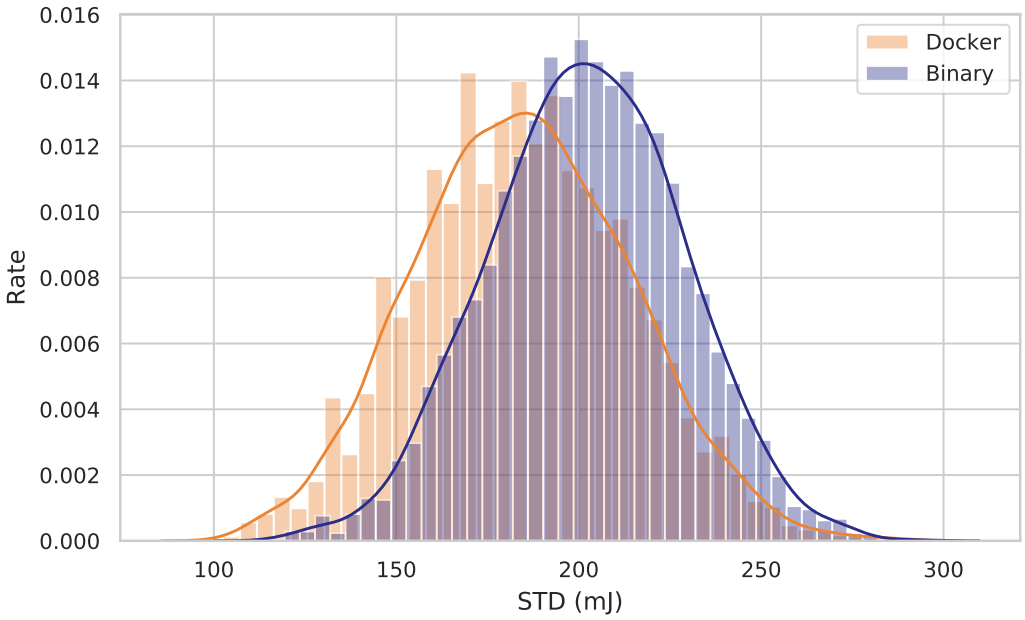

Figure 2.2: Comparing the variation of binary and Docker versions of aggregated LU, CG and EP benchmarks

2.3 Taming the energy variation for the benchmarking protocol

2.3.1 Objective

While the previous part aims to create a large umbrella that covers all empirical benchmarks related to computer science, we rather aim to narrow our study to energy benchmarking only. In other words, we will focus on pitfalls that hinders the energy claims of software. We are clearly aware of the impact of hardware on energy variations. However, we believe that there is still a room to control this energy variation for practitioners by using only parameters that can be tuned. To do so, we have inducted a set of empirical experiments using the guidelines provided by state of the art, to determine which controllable factors can reduce the variation of the energy consumption of benchmarks.

In this part we will try to answer the following Research questions.

RQ 1: Does the benchmarking protocol affect the energy variation?

RQ 2: How important is the impact of the processor features on the energy variation?

RQ 3: What is the impact of the operating system on the energy variation? and finally

RQ 4: Does the choice of the processor matter to mitigate the energy variation?

2.3.2 Experimental Setup

This section describes our detailed experimental environment, covering the clusters configurationand the benchmarks we used justifying our experimental methodology.

Measurement Context

There are three main contexts.

- Different machines with different configuration
- Different machines with the same configuration
- Same machine

To satisfy those requirements we have used the plateform Grid5000 (G5K) [10, 93],a large-scale and flexible testbed for experiment-driven research distributed accross all france. Grid5000 offers miultiple clusters composed with 4 upto 124 identical machine with different configurations for each cluster. For our experement we have considered 4 clusters. Our main creterions was the CPU Configuration. the table below 2.1, presents a description of the 4 clusters

Table 2.1: Description of clusters included in the study

Cluster	Processor	Nodes	RAM
Dahu	2× Intel Xeon Gold 6130	32	192 GiB
Chetemi	2× Intel Xeon E5-2630v4	15	768 GiB
Ecotype	2× Intel Xeon E5-2630Lv4	48	128 GiB
Paranoia	2× Intel Xeon E5-2660v2	8	128 GiB

As most of the nodes are equipped with two sockets (physical processors), we use the acronym CPU or socket to designate one of the two sockets and PU for the *processing unit*. For our study we coonsider hyper-threads as distincts **PU**

As an example, Figure 2.3 illustrates a detailed topology of a node belonging to the cluster Dahu.

Workload

Our choice for the benchmarks was based on two creterions.

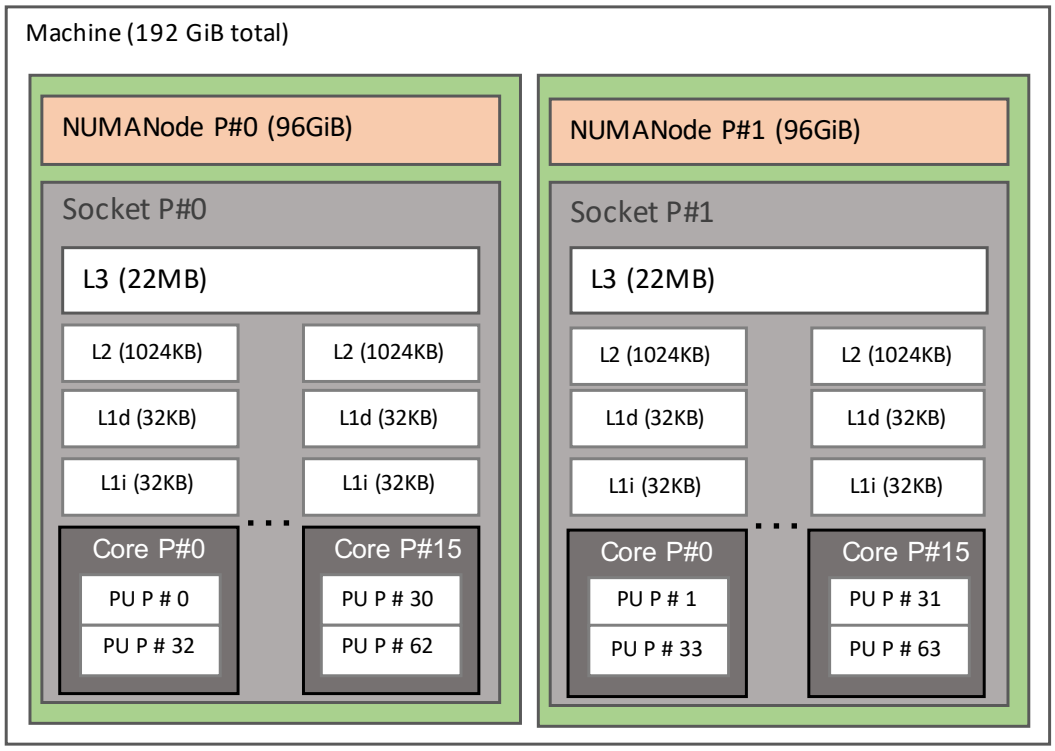

Figure 2.3: Topology of the nodes of the cluster Dahu

First, **scalability** : We wanted to gather the highest possible amount insights (regarding time spent for the experiment, therefore we wanted some benchmarks who can scale according to the number of PUs and can fulfill different scenarios. Second creterion is **representativeness** : as menstionned in the challenges, a workload has to be representative otherwise the experiment would be inconsistent [16]. To satisfy those creterions we went through state of the art and looked for the most used benchmarks for testing the performance of the hardware, and then we selected the scallable ones. Our candidate is emphNAS Parallel Benchmark (NPB v3.3.1) [8]: one of the most used benchmarks for *HPC*. We used the applications (LU), the *Conjugate Gradient* (CG) and *Embarrassingly Parallel* (EP) computation-intensive benchmarks in our experiments, with the C data class. Further more we have used other applications to validate our results using more applications such as `Stress-ng v0.10.0`,[2] `pbzip2 v1.1.9`,[3] `linpack`[4] and `sha256 v8.26`[5].

[2]https://kernel.ubuntu.com/~cking/stress-ng
[3]https://launchpad.net/pbzip2/
[4]http://www.netlib.org/linpack
[5]https://linux.die.net/man/1/sha256sum

Metrics & Measurement Tools

Our metric for the accuracy of the test is the Standard deviation aka **STD** of the energy consumption. Therefore wether the tests consumes more or less energy is out of our scope. To study this variation we need first a tool to measure the energy consumption. For this we used POWERAPI [35], which is a power monitoring toolkit that is based on *Intel Running Average Power Limit* (RAPL) [73]. The advantage of PowerAPI is that it reports the Energy consumption of CPU and DRAM at a socket level.

Our testbeds are run with a minimal version of Debian 9 (4.9.0 kernel version)[6] where we install Docker (version 18.09.5), which will be used to run the RAPL sensor and the benchmark itself. The energy sensor collects RAPL reports and stores them in a remote MONGODB instance, allowing us to perform *post-mortem* analysis in a dedicated environment. Using Docker makes the deployment process easier on the one hand, and provides us with a built-in control group encapsulation of the conducted tests on the other hand. This allows POWERAPI to measures all the running containers, even the RAPL sensor consumption, as it is isolated in a container.

Every experiment is conducted on 100 iterations, on multiple nodes and using the 3 NPB benchmarks we mentioned, with a warmup phase of 10 iterations for each experiment. In most cases, we were seeking to evaluate the *STandard Deviation* (STD), which is the most representative factor of the energy variation. We tried to be very careful, while running our experiments, not to fall in the most common benchmarking "crimes" [125]. As we study the STD difference of measurements we observed from empirical experiments, we use the bootstrap method [45] to randomly build multiple subsets of data from the original dataset, and we draw the STD density of those sets, as illustrated in Figure 2.2. Given the space constraints, this paper reports on aggregated results for nodes, benchmarks and workloads, but the raw data we collected remains available through the public repository we published.[7] We believe this can help to achieve better and more reliable comparisons. We mainly consider 3 different workloads in our experiments: single process, 50 %, and 100 %, to cover the low, medium and high CPU usage when analyzing the studied parameters effect, respectively. These workloads reflect the ratio of used PU count to the total available PU.

[6]https://github.com/grid5000/environments-recipes/blob/master/debian9-x64-min.yaml
[7]https://github.com/anonymous-data/Energy-Variation

2.3.3 Analysis

In this part, we aim to establish experimental guidelines to reduce the CPU energy variation. We therefore explore many potential factors and parameters that could have a considerable effect on the energy variation.

RQ 1: Benchmarking Protocol

To achieve a robust and reproducible experiment, practitioners often tend to repeat their tests multiple times, in order to analyze the related performance indicators, such as execution time, memory consumption or energy consumption. We therefore aim to study the benchmarking protocol to identify how to efficiently iterate the tests to capture a trustable energy consumption evaluation.

In this first experiment, we investigate if changing the testing protocol affects the energy variation. To achieve this, we considered 3 execution modes: In the "normal" mode, we iteratively run the benchmark 100 times without any extra command, while the "sleep" mode suspends the execution script for 60 seconds between iterations. Finally, the "reboot" mode automatically reboots the machine after each iteration. The difference between the normal and sleep modes intends to highlight that the CPU needs some rest before starting another iteration, especially for an intense workload. Putting the CPU into sleep for several seconds could give it some time to reach a lower frequency state or/and reduce its temperature, which could have an impact on the energy variation. The reboot mode, on the other hand, is the most straightforward way to reset the machine state after every iteration. It could also be beneficial to reset the CPU frequency and temperature, the stored data, the cache or the CPU registries. However, the reboot task takes a considerable amount of time, so rebooting the node after every single operation is not the fastest nor the most eco-friendly solution, but it deserves to be checked to investigate if it effectively enhances the overall energy variation or not.

Figure 2.4 reports on 300 aggregated executions of the benchmarks LU, CG and EP, on 4 machines of the cluster Dahu (cf. Table 2.1) for different workloads. We note that the results have been executed with different datasets sizes (B, C and D for single process, 50 % and 100 % respectively) to remedy to the brief execution times at high workloads for small datasets. This justifies the scale differences of reported energy consumptions between the 3 modes in Figure 2.4. As one can observe, picking one of these strategies does not have a strong impact on the energy variation for most workloads. In fact, all the strategies seem to exhibit the same variation with all the workloads we considered—*i.e.*, the STD is tightly close between the three modes. The only exception is the reboot mode at 100 % load, where

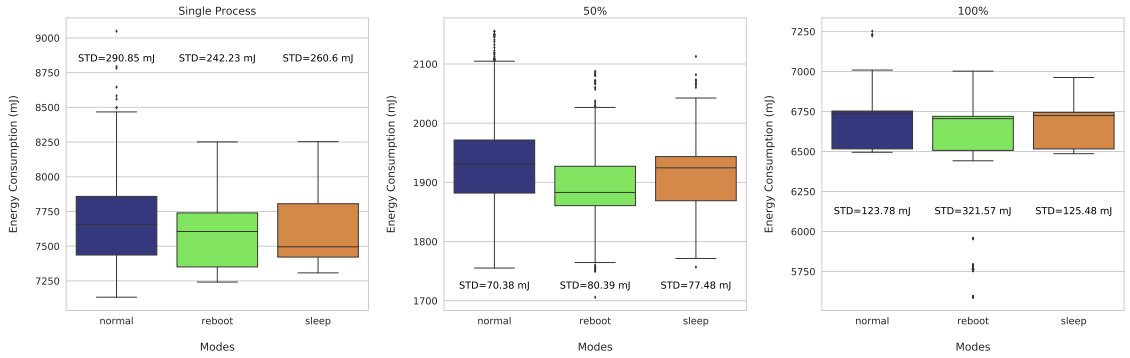

Figure 2.4: Energy variation with the normal, sleep and reboot modes

the STD is 150 % times worst, due to an important amount of outliers. This goes against our expectation, even when setting a warm-up time after reboot to stabilize the OS.

In Figure 2.5, we study the standard deviation of the three modes by constituting 5,000 random 30-iterations sets from the previous executions set and we compute the STD in each case, considering mainly the 100 % workload as the STD was 150 % higher for the reboot mode with that load. We can observe that the considerable amount of outliers in the reboot mode is not negligible, as the STD density is clearly higher than the two other modes. This makes the reboot mode as the less appropriate for the energy variation at high workloads.

To answer RQ 1, we conclude that the benchmarking protocol **partially affects** the energy variation, as highlighted by the reboot mode results for high workloads.

RQ 2: Processor Features

The C-states provide the ability to switch the CPU between more or less consuming states upon activities. Turning the C-states on or off have been subject of many discussions [127], because of its dynamic frequency mechanism but, to the best of our knowledge, there have been no fully conducted C-states behavior analysis on CPU energy variation.

We intend to investigate how much the energy consumption varies when disabling the C-states (thus, keeping the CPU in the C0 state) and at which workload. Figure 2.6 depicts the results of the experiments we executed on three nodes of the cluster Dahu. On each node, we ran the same set of benchmarks with two modes: C-states on, which is the default mode, and C-states off. Each iteration includes 100 executions of the same benchmark at a given workload, with three workload levels. We note that our results have been confirmed with the benchmarks LU, CG and EP.

We can clearly see the effect that has the C-states off mode when running a single-process application/benchmark. The energy consumption varies 5 times less than the default mode.

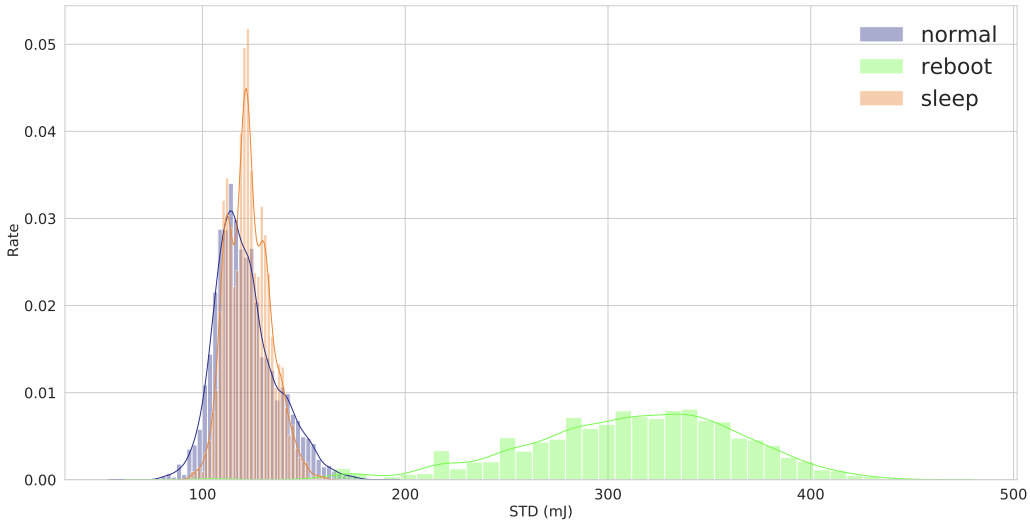

Figure 2.5: STD analysis of the normal, sleep and reboot modes

In this case, only one CPU core is used among 2×16 physical cores. The other cores are switched to a low consumption state when C-states are on, the switching operation causes an important energy consumption difference between the cores, and could be affected by other activities, such as the kernel activity, causing a notable energy consumption variation. On the other hand, switching off the C-states would keep all the cores—even the unused ones—at a high frequency usage. This highly reduces the variation, but causes up to 50 % of extra energy consumption in this test ($Mean_{C-states-off} = 11,665mJ$,$Mean_{C-states-on} = 7,641mJ$).

At a 100 % workload, disabling the C-states seems to have no effect on the total energy consumption nor its variation. In fact, all the cores are used at 100 % and the C-states module would have no effect, as the cores are not idle. The same reason would apply for the 50 % load, as the hyper-threading is active on all cores, thus causing the usage of most of them. For single process workloads, disabling the C-states causes the process to consume 50 % more energy as reported in Figure 2.6, but reduces the variation by 5 times compared to the C-states on mode. This leads to mainly two questions: Can a process pinning method reduce/increase the energy variation? And, how does the energy consumption variation evolve at different PU usage level?

Cores Pinning To answer the first question, we repeated the previous test at 50 % workload. In this experiment, we considered three cores usage strategies, the first one (S1) would pin the processes on all the PU of one of the two sockets (including hyper-threads), so it will be used

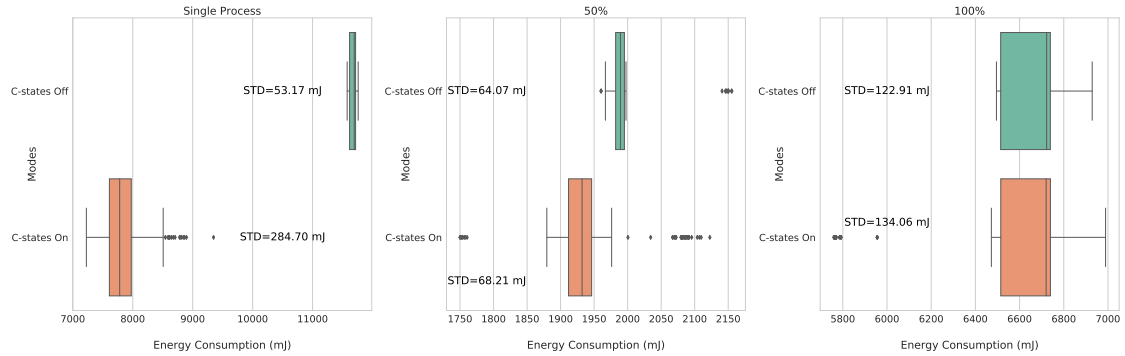

Figure 2.6: Energy variation when disabling the C-states

at 100 %, and leave the other CPU idle. The second strategy (S2) splits the workload on the two sockets so each CPU will handle 50 % of the load. In this strategy, we only use the core PU and not the hyper-threads PU, so every process would not share his core usage (all the cores are being used). The third strategy (S3) consists also on splitting the workload between the two sockets, but considering the usage of the hyper-threads on each core—*i.e.*, half of the cores are being used over the two CPU. Figure 2.7 reports on the energy consumption of the three strategies when running the benchmark `CG` on the cluster `Dahu`. We can notice the big difference between these three execution modes that we obtained only by changing the PU pinning method (that we acknowledged with more than 100 additional runs over more than 30 machines and with the benchmarks `LU` and `EP`). For example, S2 is the least power consuming strategy. We argue that the reason is related to the isolation of every process on a single physical core, reducing the context switch operations. In the first and third strategy, 32 processes are being scheduled on 16 physical cores using the hyper-threads PU, which will introduce more context switching, and thus more energy consumption.

We note that even if the first and third strategies are very similar (both use hyper-threads, but only on one CPU for the first and on two CPU for the third), the gap between them is considerable variation-wise, as the variation is 30 times lower in the first strategy ($STD_{S1} = 116mJ$,$STD_{S3} = 3,452mJ$). This shows that the usage of the hyper-threads technology is not the main reason behind the variation, the first strategy has even less variation than the second one and still uses the hyper-threading.

The reason for the S1 low energy consumption is that one of the two sockets is idle and will likely be in a lower power P-state, even with the disabled C-states. The S2 case is also low energy consuming because by distributing the threads across all the cores, it completes the task faster than in the other cases. Hence, it consumes less energy. The S3 is a high consuming strategy because both sockets are being used, but only half the cores are

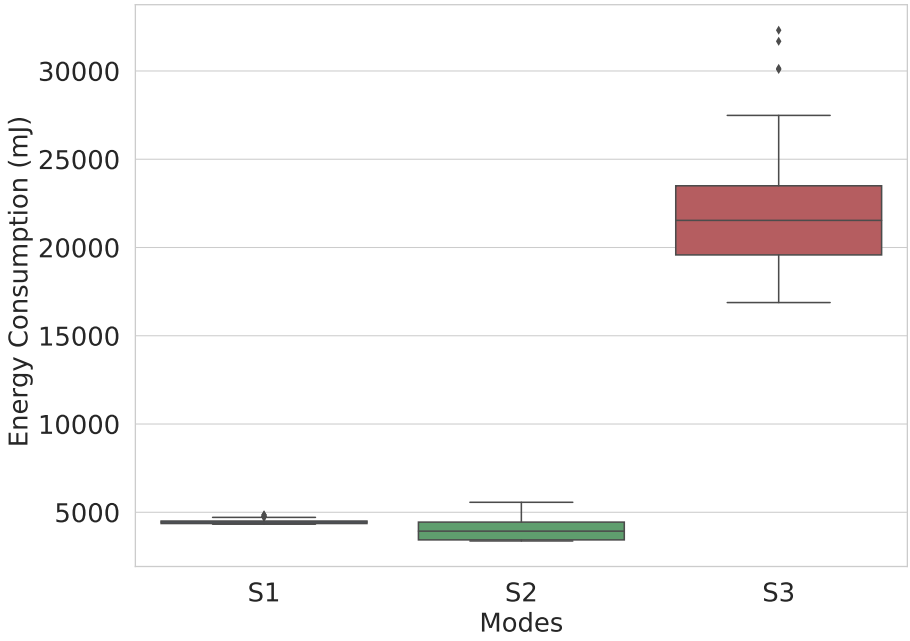

Figure 2.7: Energy variation considering the three cores pinning strategies at 50 % workload

active. This means that we pay the energy cost for both sockets being operational and for the experiments taking longer to run because of the recurrent context switching.

Our hypothesis regarding the worst results that we observed when using the third strategy is the recurrent context switching, added to the OS scheduling that could reschedule processes from a socket to another, which invalids the cache usage as a process can not take profit of the socket local L3 cache when it moves from a CPU to another (cf. Figure 2.3).

Moreover, the fact that the variation is 4–5 times higher when using the strategy S2 compared to S1 ($STD_{S1} = 116mJ$, $STD_{S3} = 575mJ$), gives another reason to believe that swapping a process from a CPU to another increases the variation due to CPU micro differences, cache misses and cache coherency. While the mean execution time for the strategy S3 is very high ($MeanTime_{S3} = 46s$) compared to the two other strategies ($MeanTime_{S1} = 11s$, $MeanTime_{S2} = 7s$), we see no correlation between the execution time and the energy variation, as the S1 still give less variations than S2 even if it takes 36 % more time to run.

Table 2.2 reports on additional aggregated results for the STD comparison on four other nodes of the cluster Dahu at 50 %, with the benchmarks LU, CG and EP. In fact, the CPU usage strategy S1 is by far the experimentation mode that gave the least variation. The STD is almost 5 times better than the strategy S2, but is up to 10 % more energy consuming ($Mean_{S1} = 4469mJ$, $Mean_{S2} = 4016mJ$). On the other hand, the strategy S3

Table 2.2: STD (mJ) comparison for 3 pinning strategies

Strategy	S1	S2	S3
Node 1	88	270	1,654
Node 2	79	283	2,096
Node 3	58	287	1,725
Node 4	51	229	1,334

is the worst, where the energy consumption can be up to 5 times higher than the strategy S2 ($Mean_{S2} = 4016mJ$, $Mean_{S3} = 21645mJ$) and the variation is much worst (30 times compared to the first strategy). These results allow us to have a better understanding of the different processes-to-PU pinning strategies, where isolating the workload on a single CPU is the best strategy. Using the hyper-threads PU on multiple sockets seems to be a bad recommendation, while keeping the hyper-threading enabled on the machine is not problematic, as long as the processes are correctly pinned on the PU. Our experiments show that running one hyper-thread per core is not always the best to do, at the opposite of the claims of [92].

Processes Threshold To answer the second question regarding the evolution of the energy variation at different levels of CPU usage, we varied the used PU's count to track the EV evolution. Figure 2.8 compares the aggregated energy variation when the C-states are on and off using 2, 4 and 8 processes for the benchmarks LU, CG and EP. This figure confirms that disabling the CPU C-states does not decrease the variation for all the workloads, as we can clearly observe, the variation is increasing along with the number of processes. When running only 2 processes, turning off the C-states reduces the STD up to 6 times, but consumes 20 % more energy ($Mean_{C-states-on} = 10,334mJ$, $Mean_{C-states-off} = 12,594mJ$). This variation is 4 times lower when running 4 processes and almost equal to the C-states on mode when running 8 processes. In fact, running more processes implies to use more CPU cores, which reduces the idle cores count, so the cores will more likely stay at a higher consumption state even if the C-states mechanism is on.

In our case, using 4 PU reduces the variation by 4 times and consumes almost the same energy as keeping the C-states mechanism on ($Mean_{C-states-on} = 7,048mJ$, $Mean_{C-states-off} = 7,119mJ$). This case would be the closest to reality as we do not want to increase the energy consumption while reducing the variation, but using a lower number of PU still results in less variation, even if it increases the overall energy consumption.

We note that disabling the C-states is not recommended in production environments, as it introduces extra energy consumption for low workloads (around 50 % in our case for a single process job). However, our goal is not to optimize the energy consumption, but to

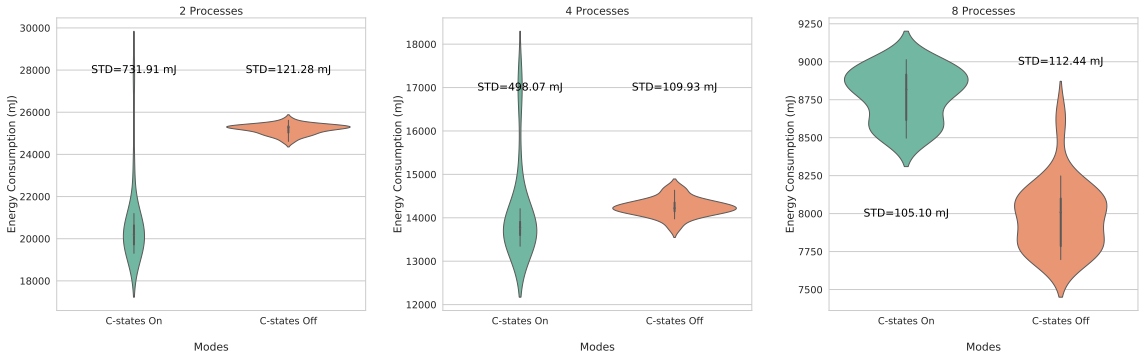

Figure 2.8: C-states effect on the energy variation, regarding the application processes count

minimize the energy variation. Thus, disabling the C-states is very important to stabilize the measurements in some cases when the variation matters the most. Comparing the energy consumptions of two algorithms or two versions of a software systems is an example of use case benefiting from this recommendation.

Turbo Boost The Turbo Boost—also known as *Dynamic Overclocking*—is a feature that has been incorporated in Intel CPU since the Sandy Bridge micro-architecture, and is now widely available on all of the Core i5, Core i7, Core i9 and Xeon series. It automatically raises some of the CPU cores operating frequency for short periods of time, and thus boost performances under specific constraints. When demanding tasks are running, the operating system decides on using the highest performance state of the processor.

Disabling or enabling the Turbo Boost has a direct impact on the CPU frequency behavior, as enabling it allows the CPU to reach higher frequencies in order to execute some tasks for a short period of time. However, its usage does not have a trivial impact on the energy variation. Acun *et al.* [4] tried to track the Turbo Boost impact on the Ivy Bridge and the Sandy Bridge architectures. They concluded that it is one of the main responsible for the energy variation, as it increases the variation from 1 % to 16 %. In our study, we included a Turbo Boost experiment in our testbed, to check this property on the recent Xeon Gold processors, covering various workloads.

The experiment we conducted showed that disabling the Turbo Boost does not exhibit any considerable positive or negative effect on the energy variation. Table 2.3 compares the STD when enabling/disabling the Turbo Boost, where the columns are a combination of workload and benchmark. In fact, we only got some minor measurements differences when switching on and off the Turbo Boost, and where in favor or against the usage of the Turbo Boost while repeating tests, considering multiple nodes and benchmarks. This behavior is mainly

Table 2.3: STD (mJ) comparison when enabling/disabling the Turbo Boost

Turbo Boost	Enabled	Disabled
EP / 5 %	310	308
CG / 25 %	95	140
LU / 25 %	204	240
EP / 50 %	84	79
EP / 100 %	125	110

related to the *thermal design power* (TDP), especially at high workloads executions. When a CPU is used at its maximum capacity, the cores would be heating up very fast and would hit the maximum TDP limit. In this case, the Turbo Boost cannot offer more power to the CPU because of the CPU thermal restrictions. At lower workloads, the tests we conducted proved that the Turbo Boost is not one of the main reasons of the energy variation. In fact, the variation difference is barely noticeable when disabling the Turbo Boost, which cannot be considered as a result regarding the OS activity and the measurement error margin. We cannot affirm that the Turbo Boost does not have an impact on all the CPU, as we only tested on two recent Xeon CPU (clusters Chetemi and Dahu). We confirmed our experiments on these machines 100 times at 5 %, 25 %, 50 % and 100 % workloads.

We conclude that CPU features **highly impact** the energy variation as an answer for RQ 2.

RQ 3: Operating System

The *operating system* (OS) is the layer that exploits the hardware capabilities efficiently. It has been designed to ease the execution of most tasks with multitasking and resource sharing. In some delicate tests and measurements, the OS activity and processes can cause a significant overhead and therefore a potential threat to the validity. The purpose behind this experiment is to determine if the sampled consumption can be reliably related to the tested application, especially for low-workload applications where CPU resources are not heavily used by the application.

The first way to do is to evaluate the OS idle activity consumption, and to compare it to a low workload running job. Therefore, we ran 100 iterations of a single process benchmark EP, LU and CG on multiple nodes from the cluster Dahu, and compared the energy behavior of the node with its idle state on the same duration. The aggregated results, illustrated in Figure 2.9, depict that the idle energy variation is up to 140 % worst than when running a job, even if it consumes 120 % less energy ($Mean_{Job} = 8,746mJ$, $Mean_{Idle} = 3,927mJ$). In fact, for the three nodes, randomly picked from the cluster Dahu, the idle variation is

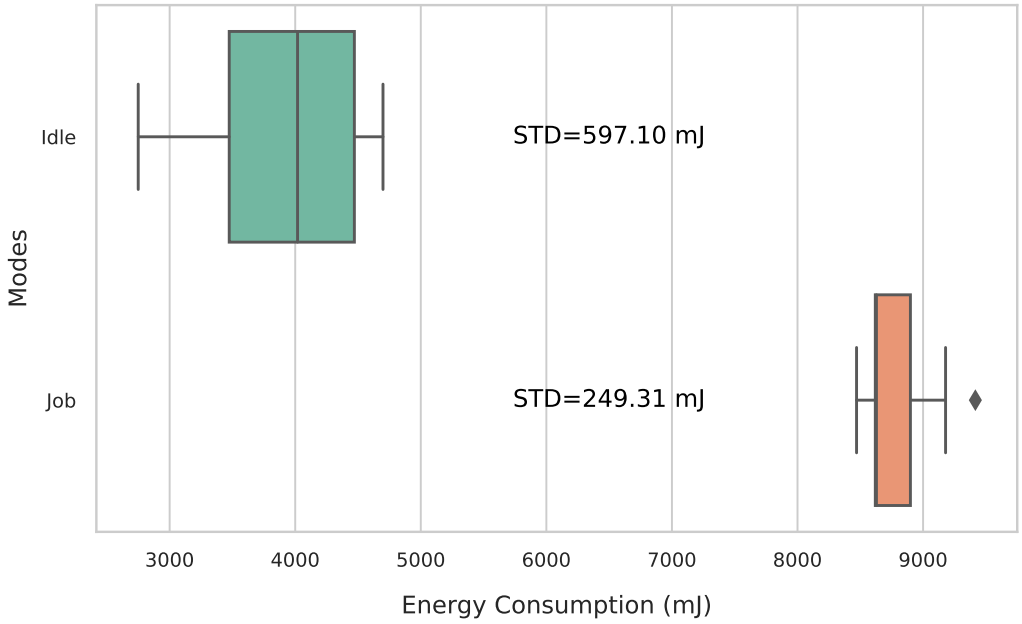

Figure 2.9: OS consumption between idle and when running a single process job

way more important than when a test was running, even if it is a single process test on a 32-cores node. This result shows that OS idle consumption varies widely, due to the lack of activity and the different CPU frequencies states, but it does not mean that this variation is the main responsible for the overall energy variation. The OS behaves differently when a job is running, even if the amount of available cores is more than enough for the OS to keep his idle behavior when running a single process.

Inspecting the OS idle energy variation is not sufficient to relate the energy variation to the active job. In fact, the OS can behave differently regarding the resource usage when running a task. To evaluate the OS and the job energy consumption separately, we used the PowerAPI toolkit. This fine-grained power meter allows the distribution of the RAPL global energy across all the Cgroups of the OS using a power model. Thus, it is possible to isolate the job energy consumption instead of the global energy consumption delivered by RAPL. To do so, we ran tests with a single process workload on the cluster Dahu, and used the PowerAPI toolkit to measure the energy consumption. Then, we compared the job energy consumption to the global RAPL data. We calculated the Pearson correlation [1] of the energy consumption and variation between global RAPL and PowerAPI, as illustrated in Figure 2.10. The job energy consumption and variation are strongly correlated with the global energy consumption and variation with the coefficients 93.6 % and 85.3 %, respectively. However, this does not completely exclude the OS activity, especially if the jobs have tight

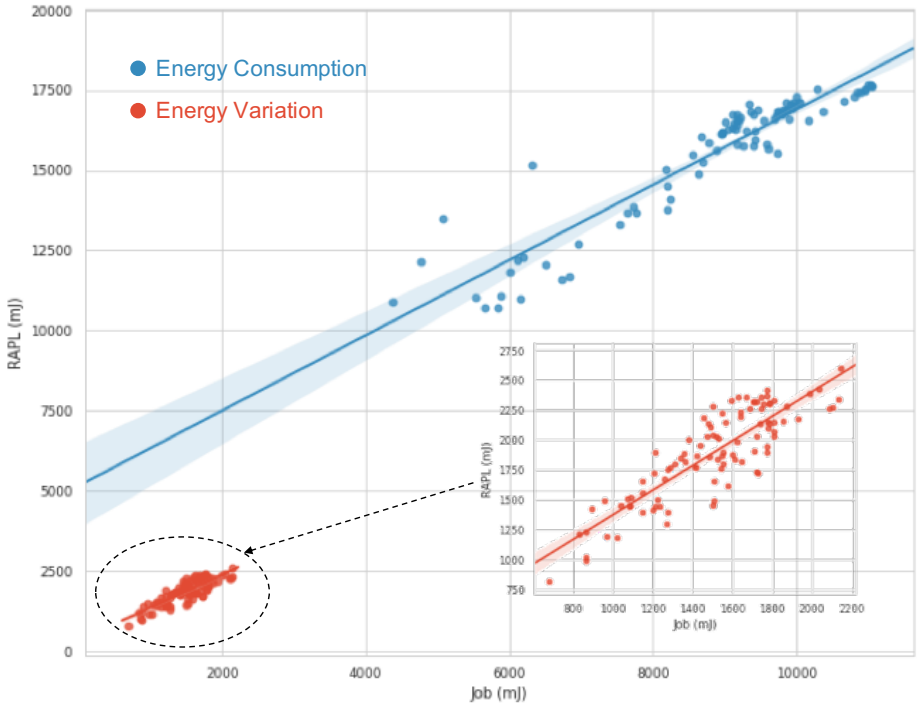

Figure 2.10: The correlation between the RAPL and the job consumption and variation

interaction with the OS through the signals and system calls. This brings a new question on whether applying extra-tuning on a minimal OS would reduce the variation? As well as what is the effect of the Meltdown security patch—that is known to be causing some performance degradation [74, 83]—on the energy variation?

OS Tuning An OS is a pack of running processes and services that might or not be required its execution. In fact, even using a minimal version of a Debian Linux, we could list many OS running services and process that could be disabled/stopped without impacting the test execution. This extra-tuning may not be the same depending on the nature of the test or the OS. Thus, we conducted a test with a deeply-tuned OS version. We disabled all the services/processes that are not essential to the OS/test running, including the OS networking interfaces and logging modules, and we only kept the strict minimum required to the experiment's execution. Table 2.4 reports on the aggregated results for running single process measurements with the benchmarks CG, LU and EP, on three servers of the cluster Dahu, before and after tuning the OS. Every cell contains the *STD* value before the tuning, plus/minus a ratio of the energy variation after the tuning. We notice that the energy variation varies less than 10 % after the extra-tuning. We argue that this variation is not substantial, as it is not stable from a node to another. Moreover, 10 % of variation is not a representative

Table 2.4: STD (mJ) comparison before/after tuning the OS

Node	EP	CG	LU
N1	1370 -9 %	78 +7 %	128 +2 %
N2	1278 -7 %	64 -1 %	120 +9 %
N3	1118 +1 %	83 +2 %	93 +7 %

difference, due to many factors that can affect it as the CPU temperature or the measurement errors.

Speculative Executions Meltdown and Spectre are two of the most famous hardware vulnerabilities discovered in 2018, and exploiting them allows a malicious process to access others processes data that is supposed to be private [74, 83]. They both exploit the speculative execution technique where a process anticipates some upcoming tasks, which are not guaranteed to be executed, when extra resources are available, and revert those changes if not. Some OS-level patches had been applied to prevent/reduce the criticality of these vulnerabilities. On the Linux kernel, the patch has been automatically applied since the version 4.14.12. It mitigates the risk by isolating the kernel and the user space and preventing the mapping of most of the kernel memory in the user space. Nikolay *et al.* have studied in [120] the impact of patching the OS on the performance. The results showed that the overall performance decrease is around 2–3 % for most of the benchmarks and real-world applications, only some specific functions can meet a high performance decrease. In our study, we are interested in the applied patch's impact on the energy variation, as the performance decrease could mean an energy consumption increase. Thus, we ran the same benchmarks LU, CG ad EP on the cluster Dahu with different workloads, using the same OS, with and without the security patch. Table 2.5 reports on the STD values before disabling the security patch. A minus means that the energy varies less without the patch being applied, while a plus means that it varies more. These results help us to conclude that the security patch's effect on the energy variation is not substantial and can be absorbed through the error margin for the tested benchmarks. In fact, the best case to consider is the benchmark LU where the energy variation is less than 10 % when we disable the security patch, but this difference is still moderate. The little performance difference discussed in [74, 83] may only be responsible of a small variation, which will be absorbed through the measurement tools and external noise error margin in most cases.

To answer RQ 3, we conclude that the OS **should not be the main focus** of the energy variation taming efforts.

Table 2.5: STD (mJ) comparison with/without the security patch

Node	EP	CG	LU
N1	269 +2 %	83 +1 %	108 -6 %
N2	195 +1 %	84 -5 %	121 -9 %
N3	223 +/-1 %	72 -4 %	117 +8 %
N4	276 +3 %	60 +0 %	113 -3 %

Table 2.6: STD (mJ) comparison of experiments from 4 clusters

Cluster	**Dahu**	**Chetemi**	**Ecotype**	**Paranoia**
Arch	Skylake	Broadwell	Broadwell	Ivy Bridge
Freq	3.7 GHz	3.1 GHz	2.9 GHz	3.0 GHz
TDP	125 W	85 W	55 W	95 W
5%	364	210	**75**	**76**
50%	98	86	**49**	244
100%	119	116	**106**	240

RQ 4: Processor Generation

Intel microprocessors have noticeably evolved during these last 20 years. Most of the new CPU come with new enhancements to the chip density, the maximum Frequency or some optimization features like the C-states or the Turbo Boost. This active evolution caused that different generations of CPU can handle a task differently. The aim of this expriment is not to justify the evolution of the variation across CPU versions/generations, but to observe if the user can choose the best node to execute her experiments. Previous papers have discussed the evolution of the energy consumption variation across CPU generations and concluded that the variation is getting higher with the latest CPU generations [Wang et al., 92], which makes measurements stability even worse. In this experiment, we therefore compare four different generations of CPU with the aim to evaluate the energy variation for each CPU and its correlation with the generation. Table 2.6 indicates the characteristics of each of the tested CPU.

Table 2.6 also shows the aggregated energy variation of the different generations of nodes for the benchmarks LU, CG and EP. The results attest that the latest versions of CPU do not necessarily cause more variation. In the experiments we ran, the nodes from the cluster Paranoia tend to cause more variation at high workloads, even if they are from the latest generation. While the Skylake CPU of the cluster Dahu cause often more energy variation than Chetemi and the Ecotype Broadwell CPU. We argue that the hypothesis "*the energy consumption on newer CPU varies more*" could be true or not depending on the compared generations, but most importantly, the chips energy behaviors. On the other hand,

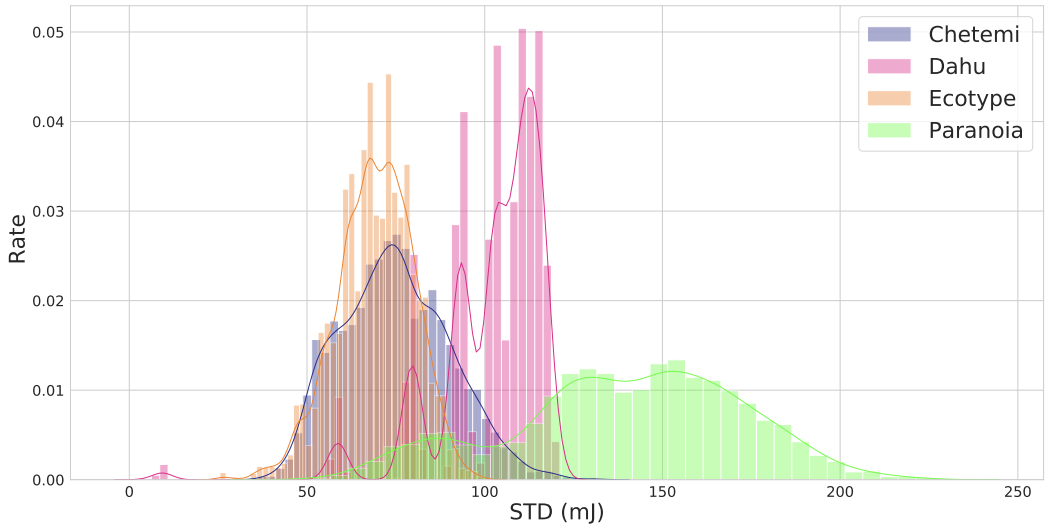

Figure 2.11: Energy consumption STD density of the 4 clusters

our experiments showed the lowest energy variation when using the Ecotype CPU, these CPU are not the oldest nor the latest, but are tagged with "L" for their low power/TDP. This result rises another hypothesis when considering CPU choice, which implies selecting the CPU with a low TDP. This hypothesis has been confirmed on all the Ecotype cluster nodes, especially at low and medium workloads.

Figure 2.11 is an illustration of the aggregated STD density of more than 5,000-random values sets taken from all the conducted experiments. This shows that the cluster Paranoia reports the worst variation in most cases, and that Ecotype is the best cluster to consider to get the least variations, as it has a higher density for small variation values.

We conclude on **affirming RQ 4**, as selecting the right CPU can help to get less variations.

2.3.4 Experimental Guidelines

To summarize our experiments, we provide some experimental guidelines in Table 2.7, based on the multiple experiments and analysis we did. These guidelines constitute a set of minimal requirements or best practices, depending on the workload and the criticality of the energy measurement precision. It therefore intends to help practitioners in taming the energy variation on the selected CPU, and conduct the experiments with the least variations.

Table 2.7: Experimental Guidelines for Energy Variations

Guideline	**Load**	**Gain**
Use a low TDP CPU	Low & medium	Up to 3×
Disable the CPU C-states	Low	Up to 6×
Use the least of sockets in a case of multiple CPU	Medium	Up to 30×
Avoid the usage of hyper-threading whenever possible	Medium	Up to 5×
Avoid rebooting the machine between tests	High	Up to 1.5×
Do not relate to the machine idle variation to isolate a test EC, the CPU/OS changes its behavior when a test is running and can exhibit less variation than idle	Any	—
Rather focus the optimization efforts on the system under test than the OS	Any	—
Execute all the similar and comparable experiments on a same machine. Identical machines can exhibit many differences regarding their energy behavior	Any	Up to 1.3×

Table 2.7 gives a proper understanding of known factors, like the C-states and its variation reduction at low workloads. However, it also lists some new factors that we identified along the analysis we conducted in Section, such as the results related to the OS or the reboot mode. Some of the guidelines are more useful/efficient for specific workloads, as showed in our experiments. Thus, qualifying the workload before conducting the experiments can help in choosing the proper guidelines to apply. Other studied factors are not been mentioned in the guidelines, like the Turbo Boost or the Speculative execution, due to the small effect that has been observed in our study.

In order to validate the accuracy of our guidelines among a varied set of benchmarks on one hand, and their effect on the variation between identical machines on the other hand, we ran seven experiments with benchmarks and real applications on a set of four identical nodes from the cluster Dahu, before (normal mode where everything is left to default and to the charge of the OS) and after (optimized) applying our guidelines. Half of these experiments has been performed at a 50 % workload and the other half on single process jobs. The choice of these two workloads is related to the optimization guidelines that are mainly effective at

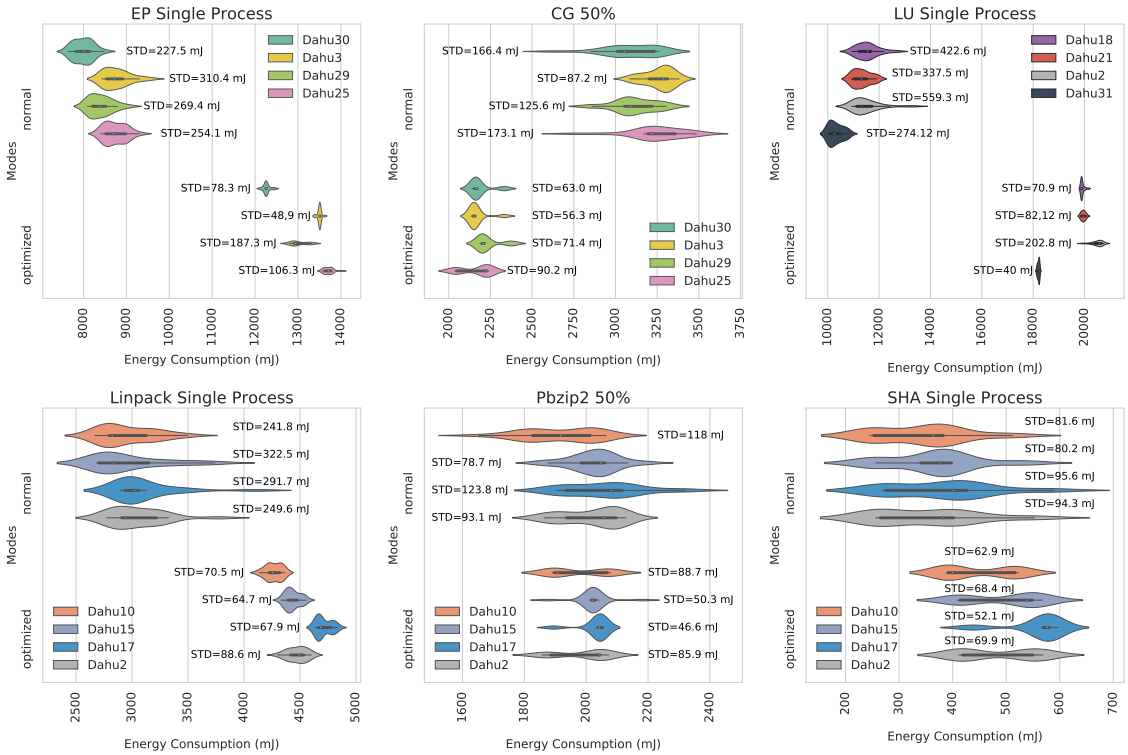

Figure 2.12: Energy variation comparison with/without applying our guidelines

low and medium workloads. We note that we used the cluster Dahu over Ecotype to highlight the guidelines effect on the nodes where the variation is susceptible to be higher.

Figure 2.12 and 2.13 highlight the improvement brought by the adoption of our guidelines. They demonstrate the intra-node STD reduction at low and medium workloads for all the benchmarks used at different levels. Concretely, for low workloads, the energy variation is 2–6 times lower after applying the optimization guidelines for the benchmarks LU and EP, as well as LINPACK, while it is 1.2–1.8 times better for `Sha256`. For this workload, the overall energy consumption after optimization can be up to 80 % higher due to disabling the C-states to keep all the unused cores at a high power consumption state ($Mean_{LU-normal-Dahu2} = 11,500mJ$, $Mean_{LU-optimized-Dahu2} = 20,508mJ$). For medium workloads, the STD, and thus variation, is up to 100 % better for the benchmark CG, 20–150 % better for the `pbzip2` application and up to 100% for STRESS-NG. We note that the optimized version consumes less energy thanks to an appropriate core pinning method.

Figures 2.12 and 2.13 also highlight that applying the guidelines does not reduce the inter-nodes variation in all the cases. This variation can be up to 30 % in modern CPU [Wang et al.].

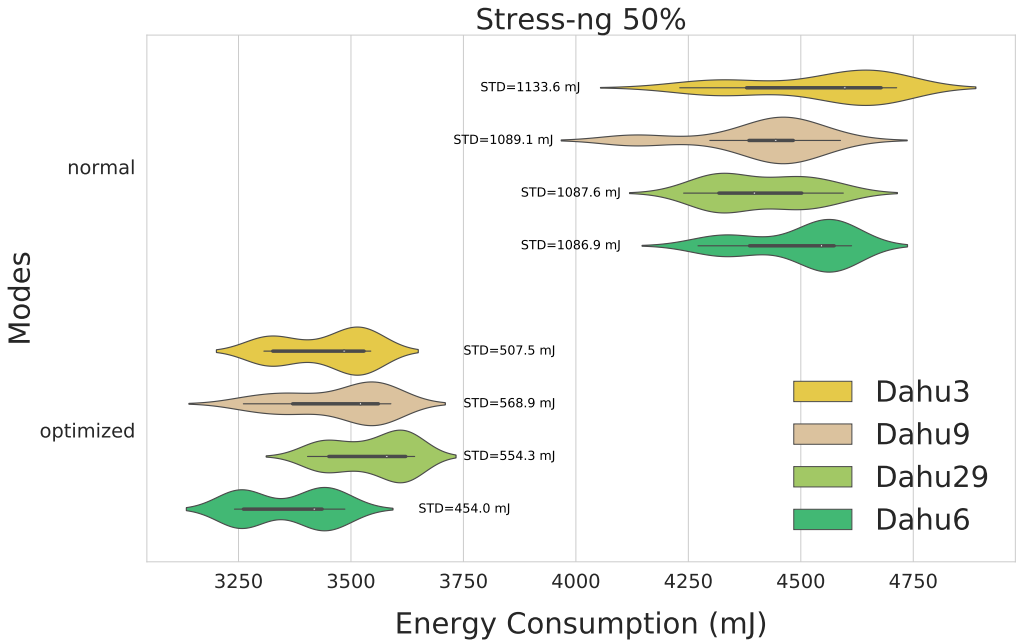

Figure 2.13: Energy variation comparison with/without applying our guidelines for STRESS-NG

However, taming the intra-node variation is a good strategy to identify more relevant mediums and medians, and then perform accurate comparisons between the nodes variation. Even though, using the same node is always better, to avoid the extra inter-nodes variation and thus improve the stability of measurements.

2.3.5 Threats to Validity

A number of issues affect the validity of our work. For most of our experiments, we used the Intel RAPL tool, which has evolved along Intel CPU generations to be known as one of the most accurate tools for modern CPU, but still adds an important overhead if we adopt a sampling at high frequency. The other fine-grained tool we used for measurements is POWERAPI. It allows to measure the energy consumption at the granularity of a process or a Cgroup by dividing the RAPL global energy over the running processes using a power model. The usage of POWERAPI adds an error margin because of the power model built over RAPL. The RAPL tool mainly measures the CPU and DRAM energy consumption. However, even running CPU/RAM intensive benchmarks would keep a degree on uncertainty concerning the hard disk and networking energy consumption. In addition, the operating system adds a layer of confusion and uncertainty.

The Intel CPU chip manufacturing process and the materials micro-heterogeneity is one of the biggest issues, as we cannot track or justify some of the energy variation between identical CPU or cores. These CPU/cores might handle frequencies and temperature differently and behave consequently. This hardware heterogeneity also makes reproduction complex and requires the usage of the same nodes on the cluster with the same OS.

2.3.6 Conclusion

In this part, we conducted an empirical study of controllable factors that can increase the energy variations on platforms with some of the latest CPU, and for several workloads. We provide a set of guidelines that can be implemented and tuned (through the OS GRUB for example), especially with the new data centers isolation trend and the cloud usage, even for scientific and R&D purposes. Our guidelines aim at helping the user in reducing the CPU energy variation during systems benchmarking, and conduct more stable experiments when the variation is critical. For example, when comparing the energy consumption of two versions of an algorithm or a software system, where the difference can be tight and need to be measured accurately.

Overall, our results are not intended to nullify the variability of the CPU, as some of this variability is related to the chip manufacturing process and its thermal behavior. The aim of our work is to be able to tame and mitigate this variability along controlled experiments. We studied some previously discussed aspects on some recent CPU, considered new factors that have not been deeply analyzed to the best of our knowledge, and constituted a set of guidelines to achieve the variability mitigating purpose. Some of these factors, like the C-states usage, can reduce the energy variation up to 500 % at low workloads, while choosing the wrong cores/PU strategy can cause up to 30$\times$ more variability.

We believe that our approach can also be used to study/discover other potential variability factors, and extend our results to alternative CPU generations/brands. Most importantly, this should motivate future works on creating a better knowledge on the variability due to CPU manufacturing process and other factors.

2.4 Perspectives

By the end of this study we have gathered enought guidelines to make the tests more reproducible, accurate. We created a set of new tests named **energy tests** which are more similar to performance tests. Thanks to the work of two interns [mamadou and adrien] we created a CI/CD plateform to measure the energy consumption of Java projects and we could

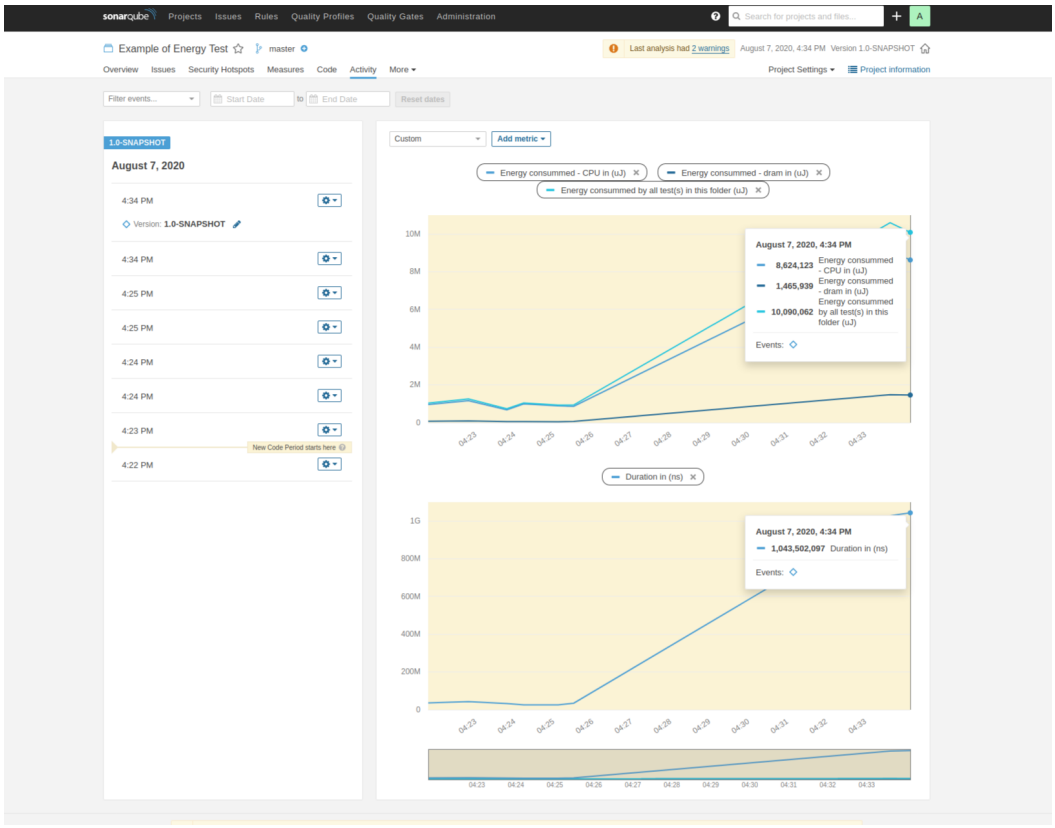

Figure 2.14: Example of the Junit Sonar Plugin

track the evolution of the this energy accross different stages of the project. In the figure below we see an example of this plugin. For more details please visit the gitlab repository ... add link.

Bibliography

[1] (2008). *Pearson's Correlation Coefficient*. Springer Netherlands.

[2] Abuabdo, A. and Al-Sharif, Z. A. (2019). Virtualization vs. Containerization: Towards a Multithreaded Performance Evaluation Approach. In *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, pages 1–6.

[3] Acun, B., Miller, P., and Kale, L. V. (2016a). Variation among processors under turbo boost in hpc systems. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–12.

[4] Acun, B., Miller, P., and Kale, L. V. (2016b). Variation Among Processors Under Turbo Boost in HPC Systems.

[5] Akeret, J., Gamper, L., Amara, A., and Refregier, A. (2015). HOPE: A Python just-in-time compiler for astrophysical computations. *Astronomy and Computing*, 10:1–8.

[6] Arcuri, A. and Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceeding of the 33rd International Conference on Software Engineering - ICSE '11*, page 1, Waikiki, Honolulu, HI, USA. ACM Press.

[Bailey et al.] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. The NAS Parallel Benchmarks. page 79.

[8] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K. (1991). The nas parallel benchmarks—summary and preliminary results.

[9] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. (2009). Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 280–293.

[10] Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., and Sarzyniec, L. (2013). Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*. Springer.

[11] Banerjee, A. and Roychoudhury, A. (2016). Automated re-factoring of android apps to enhance energy-efficiency. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, pages 139–150.

[12] Baskar, P., Joseph, M. A., Narayanan, N., and Loya, R. B. (2013). Experimental investigation of oxygen enrichment on performance of twin cylinder diesel engine with variation of injection pressure. In *2013 International Conference on Energy Efficient Technologies for Sustainability*, pages 682–687. IEEE.

[13] Bedard, D., Lim, M. Y., Fowler, R., and Porterfield, A. (2010). Powermon: Fine-grained and integrated power monitoring for commodity computer systems. In *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, pages 479–484. IEEE.

[14] Ben Asher, Y. and Rotem, N. (2009). The effect of unrolling and inlining for Python bytecode optimizations. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference on - SYSTOR '09*, page 1, Haifa, Israel. ACM Press.

[15] Benmoussa, K., Laaziri, M., Khoulji, S., Larbi, K. M., and Yamami, A. E. (2019). A new model for the selection of web development frameworks: Application to PHP frameworks. *International Journal of Electrical and Computer Engineering (IJECE)*, 9(1):695–703.

[16] Blackburn, S. M., Diwan, A., Hauswirth, M., Sweeney, P. F., Nelson Amaral, J., Tuma, P., Pankratius, V., Nystrom, N., Moret, P., Kalibera, T., and et al. (2012). Evaluate collaboratory technical report: Can you trust your experimental results?

[17] Borkar, S. (2005). Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6).

[18] Breitbart, J., Weidendorfer, J., and Trinitis, C. (2015). Case study on co-scheduling for hpc applications. In *2015 44th International Conference on Parallel Processing Workshops*, pages 277–285. IEEE.

[19] Bujnowski, G. and Smołka, J. (2020). Java and kotlin code performance in selected web frameworks. *Journal of Computer Sciences Institute*, 16:219–226.

[20] Bukh, P. N. D. (1992). The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling.

[21] Burtscher, M., Zecena, I., and Zong, Z. (2014). Measuring gpu power with the k20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*, pages 28–36.

[Buytaert and Eeckhout] Buytaert, A. G. D. and Eeckhout, L. Statistically Rigorous Java Performance Evaluation. page 20.

[Caldeira et al.] Caldeira, A. B., Grabowski, B., Haug, V., Kahle, M.-E., Laidlaw, A., Maciel, C. D., Sanchez, M., and Sung, S. Y. Ibm power system s822.

[24] Caldeira, A. B., Grabowski, B., Haug, V., Kahle, M.-E., Maciel, C. D., and Sanchez, M. (2014). Ibm power systems s814 and s824 technical overview and introduction. *IBM Redbook REDP-5097-00*.

[25] Calero, C., Mancebo Pavón, J., and García, F. (2021). Does maintainability relate to the energy consumption of software?

[26] Cardenas, L. G., Gil, J. A., Domenech, J., Sahuquillo, J., and Pont, A. (2005). Performance comparison of a Web cache simulation framework. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA Papers)*, volume 2, pages 281–284 vol.2.

[27] Chamas, C. L., Cordeiro, D., and Eler, M. M. (2017). Comparing rest, soap, socket and grpc in computation offloading of mobile applications: An energy cost analysis. In *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE.

[28] Chasapis, D., Casas, M., Moretó, M., Schulz, M., Ayguadé, E., Labarta, J., and Valero, M. (2016a). Runtime-guided mitigation of manufacturing variability in power-constrained multi-socket numa nodes. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–12.

[29] Chasapis, D., Schulz, M., Casas, M., Ayguadé, E., Valero, M., Moretó, M., and Labarta, J. (2016b). Runtime-Guided Mitigation of Manufacturing Variability in Power-Constrained Multi-Socket NUMA Nodes.

[30] Chiba, T., Yoshimura, T., Horie, M., and Horii, H. (2018). Towards selecting best combination of sql-on-hadoop systems and jvms. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 245–252. IEEE.

[31] Coles, H., Qin, Y., and Price, P. (2014a). Comparing Server Energy Use and Efficiency Using Small Sample Sizes. Technical Report LBNL-6831E, 1163229.

[32] Coles, H. C., Qin, Y., and Price, P. N. (2014b). Comparing server energy use and efficiency using small sample sizes. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[33] Coley, G. (2012). Beaglebone rev a6 system reference manual. *Obtenido*, 4(23):2012.

[34] Colmant, M., Rouvoy, R., Kurpicz, M., Sobe, A., Felber, P., and Seinturier, L. (2018a). The next 700 cpu power models. *Journal of Systems and Software*, 144:382–396.

[35] Colmant, M., Rouvoy, R., Kurpicz, M., Sobe, A., Felber, P., and Seinturier, L. (2018b). The next 700 CPU power models. *Journal of Systems and Software*, 144.

[36] Corral, L., Georgiev, A. B., Sillitti, A., and Succi, G. (2014). Method reallocation to reduce energy consumption: an implementation in android os. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1213–1218.

[37] Couto, M., Pereira, R., Ribeiro, F., Rua, R., and Saraiva, J. (2017). Towards a green ranking for programming languages. In *Proceedings of the 21st Brazilian Symposium on Programming Languages*, pages 1–8.

[38] Crist, J. (2016). Dask Numba: Simple libraries for optimizing scientific python code. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2342–2343.

[39] Das, R. and Saikia, D. L. P. (2016). Comparison of Procedural PHP with Codeigniter and Laravel Framework. /paper/A-new-model-for-the-selection-of-web-development-to-Benmoussa-Laaziri/6eb6977a37b2d38e232472b43897ab6047bd751c.

[40] de Matos, F. F. S., Rego, P. A., and Trinta, F. A. M. (2021). An empirical study about the adoption of multi-language technique in computation offloading in a mobile cloud computing scenario. In *CLOSER*, pages 207–214.

[41] Destefanis, G., Ortu, M., Porru, S., Swift, S., and Marchesi, M. (2016). A Statistical Comparison of Java and Python Software Metric Properties. In *Proceedings of the 7th International Workshop on Emerging Trends in Software Metrics*, WETSoM '16, pages 22–28, New York, NY, USA. ACM.

[42] Diouri, M. E. M., Glück, O., Lefevre, L., and Mignot, J.-C. (2013). Your cluster is not power homogeneous: Take care when designing green schedulers! In *2013 International Green Computing Conference Proceedings*, pages 1–10. IEEE.

[43] Echtler, F. and Häußler, M. (2018). Open source, open science, and the replication crisis in hci. association for computing machinery, new york, ny, usa, 1–8.

[44] Eddie Antonio Santos, Carson McLean, Christophr Solinas, and Abram Hindle (2017). How does docker affect energy consumption? Evaluating workloads in and out of Docker containers. *The journal of systems & Software*.

[45] Efron, B. (2000). The bootstrap and modern statistics. *Journal of the American Statistical Association*, 95(452).

[46] El Mehdi Diouri, M., Gluck, O., Lefevre, L., and Mignot, J.-C. (2013). Your cluster is not power homogeneous: Take care when designing green schedulers!

[47] Fahad, M., Shahid, A., Manumachu, R. R., and Lastovetsky, A. (2019). A comparative study of methods for measurement of energy of computing. *Energies*, 12(11):2204.

[48] Fernandes, B., Pinto, G., and Castor, F. (2017). Assisting non-specialist developers to build energy-efficient software. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 158–160. IEEE.

[49] Fieni, G., Rouvoy, R., and Seinturier, L. (2020). Smartwatts: Self-calibrating software-defined power meter for containers. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 479–488. IEEE.

[50] Fieni, G., Rouvoy, R., and Seiturier, L. (2021). Selfwatts: On-the-fly selection of performance events to optimize software-defined power meters. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 324–333. IEEE.

[51] Gajewski, M. and Zabierowski, W. (2019). Analysis and Comparison of the Spring Framework and Play Framework Performance, Used to Create Web Applications in Java. In *2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 170–173.

[52] Ge, R., Feng, X., Song, S., Chang, H.-C., Li, D., and Cameron, K. W. (2009). Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671.

[53] Goodman, S. N., Fanelli, D., and Ioannidis, J. P. (2016). What does research reproducibility mean? *Science translational medicine*, 8(341):341ps12–341ps12.

[54] Grambow, M., Meusel, L., Wittern, E., and Bermbach, D. (2020). Benchmarking microservice performance: A pattern-based approach. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 232–241, Brno Czech Republic. ACM.

[55] Guimarães, M., Saraiva, J., and Belo, O. (2016). Some heuristic approaches for reducing energy consumption on database systems. *DBKDA 2016*, page 59.

[56] Hackenberg, D., Ilsche, T., Schöne, R., Molka, D., Schmidt, M., and Nagel, W. E. (2013). Power measurement techniques on standard compute nodes: A quantitative comparison. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 194–204. IEEE.

[57] Hackenberg, D., Ilsche, T., Schuchart, J., Schöne, R., Nagel, W. E., Simon, M., and Georgiou, Y. (2014). Hdeem: high definition energy efficiency monitoring. In *2014 Energy Efficient Supercomputing Workshop*, pages 1–10. IEEE.

[58] Hackenberg, D., Schöne, R., Ilsche, T., Molka, D., Schuchart, J., and Geyer, R. (2015). An energy efficiency feature survey of the intel haswell processor. In *2015 IEEE international parallel and distributed processing symposium workshop*, pages 896–904. IEEE.

[59] Hammouda, A., Siegel, A. R., and Siegel, S. F. (2015). Noise-Tolerant Explicit Stencil Computations for Nonuniform Process Execution Rates. *ACM Transactions on Parallel Computing*, 2(1):1–33.

[60] Hasan, S., King, Z., Hafiz, M., Sayagh, M., Adams, B., and Hindle, A. (2016). Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering*, pages 225–236.

[61] He, S., Manns, G., Saunders, J., Wang, W., Pollock, L., and Soffa, M. L. (2019). A statistics-based performance testing methodology for cloud applications. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, pages 188–199, Tallinn, Estonia. ACM Press.

[62] Heinrich, F., Carpen-Amarie, A., Degomme, A., Hunold, S., Legrand, A., Orgerie, A.-C., and Quinson, M. (2017a). Predicting the performance and the power consumption of mpi applications with simgrid.

[63] Heinrich, F., Carpen-Amarie, A., Degomme, A., Hunold, S., Legrand, A., Orgerie, A.-C., and Quinson, M. (2017b). Predicting the Performance and the Power Consumption of MPI Applications With SimGrid.

[64] Hindle, A., Wilson, A., Rasmussen, K., Barlow, E. J., Campbell, J. C., and Romansky, S. (2014). Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th working conference on mining software repositories*, pages 12–21.

[65] Hirst, J. M., Miller, J. R., Kaplan, B. A., and Reed, D. D. (2013). Watts up? pro ac power meter for automated energy recording.

[66] Howe, B. (2012). Virtual Appliances, Cloud Computing, and Reproducible Research. *Computing in Science Engineering*, 14(4):36–41.

[67] Ilsche, T., Hackenberg, D., Graul, S., Schone, R., and Schuchart, J. (2015). Power measurements for compute nodes: Improving sampling rates, granularity and accuracy. In *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, Las Vegas, NV, USA. IEEE.

[68] Inadomi, Y., Patki, T., Inoue, K., Aoyagi, M., Rountree, B., Schulz, M., Lowenthal, D., Wada, Y., Fukazawa, K., Ueda, M., et al. (2015a). Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE.

[69] Inadomi, Y., Ueda, M., Kondo, M., Miyoshi, I., Patki, T., Inoue, K., Aoyagi, M., Rountree, B., Schulz, M., Lowenthal, D., Wada, Y., and Fukazawa, K. (2015b). Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing.

[70] Islam, S., Noureddine, A., and Bashroush, R. (2016). Measuring energy footprint of software features. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–4. IEEE.

[71] Jagroep, E., Procaccianti, G., van der Werf, J. M., Brinkkemper, S., Blom, L., and van Vliet, R. (2017). Energy efficiency on the product roadmap: an empirical study across releases of a software product. *Journal of Software: Evolution and process*, 29(2):e1852.

[72] Joakim v Kisroski, Hansfreid Block, John Beckett, Cloyce Spradling, Klaus-Dieter Lange, and Samuel Kounev (2016). Variations in CPU Power Consumption.

[73] Khan, K. N., Hirki, M., Niemi, T., Nurminen, J. K., and Ou, Z. (2018). Rapl in action: Experiences in using rapl for power measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 3(2).

[74] Kocher, P., Horn, J., Fogh, A., , Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. (2019). Spectre attacks: Exploiting speculative execution.

[75] Koomey, J. et al. (2011). Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, 9(2011):161.

[76] Kothari, N. and Bhattacharya, A. (2009). Joulemeter: Virtual machine power measurement and management. *MSR Tech Report*.

[77] Kurpicz, M., Orgerie, A.-C., and Sobe, A. (2016). How much does a vm cost? energy-proportional accounting in vm-based environments. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 651–658. IEEE.

[78] Lafond, S. and Lilius, J. (2006). An energy consumption model for an embedded java virtual machine. In *International Conference on Architecture of Computing Systems*, pages 311–325. Springer.

[79] Laros, J. H., Pokorny, P., and DeBonis, D. (2013). Powerinsight-a commodity power measurement capability. In *2013 International Green Computing Conference Proceedings*, pages 1–6. IEEE.

[80] LeBeane, M., Ryoo, J. H., Panda, R., and John, L. K. (2015). Watt watcher: fine-grained power estimation for emerging workloads. In *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 106–113. IEEE.

[81] Li, Y. and Schwiebert, L. (2016). Boosting Python Performance on Intel Processors: A Case Study of Optimizing Music Recognition. In *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*, pages 52–58.

[82] Lilja, D. J. (2005). *Measuring computer performance: a practitioner's guide*. Cambridge university press.

[83] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. (2018). Meltdown: Reading kernel memory from user space.

[84] Liu, K., Pinto, G., and Liu, Y. D. (2015). Data-oriented characterization of application-level energy optimization. In *International Conference on Fundamental Approaches to Software Engineering*, pages 316–331. Springer.

[85] Longo, M., Rodriguez, A. V., Mateos Diaz, C. M., and Zunino Suarez, A. O. (2019). Reducing energy usage in resource-intensive java-based scientific applications via micro-benchmark based code refactorings.

[86] Lovicott, D. (2009). Thermal design of the dell™ poweredge™ t610™, r610™, and r710™ servers. *Round Rock. Texas*.

[87] Ma, H., Simon, D., Siarry, P., Yang, Z., and Fei, M. (2017). Biogeography-based optimization: a 10-year review. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(5):391–407.

[88] Manotas, I., Pollock, L., and Clause, J. (2014). Seeds: A software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering*, pages 503–514.

[89] Manotas, I., Sahin, C., Clause, J., Pollock, L., and Winbladh, K. (2013a). Investigating the impacts of web servers on web application energy usage. In *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*, pages 16–23. IEEE.

[90] Manotas, I., Sahin, C., Clause, J., Pollock, L., and Winbladh, K. (2013b). Investigating the impacts of web servers on web application energy usage. In *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*, pages 16–23.

[91] Marathe, A., Zhang, Y., Blanks, G., Kumbhare, N., Abdulla, G., and Rountree, B. (2017a). An empirical survey of performance and energy efficiency variation on intel processors. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, pages 1–8.

[92] Marathe, A., Zhang, Y., Blanks, G., Kumbhare, N., Abdulla, G., and Rountree, B. (2017b). An empirical survey of performance and energy efficiency variation on Intel processors.

[93] Margery, D., Morel, E., Nussbaum, L., Richard, O., and Rohr, C. (2014). Resources Description, Selection, Reservation and Verification on a Large-scale Testbed.

[94] McCreary, H.-Y., Broyles, M. A., Floyd, M. S., Geissler, A. J., Hartman, S. P., Rawson, F. L., Rosedahl, T. J., Rubio, J. C., and Ware, M. S. (2007). Energyscale for ibm power6 microprocessor-based systems. *IBM Journal of Research and Development*, 51(6):775–786.

[95] Mirowski, P., Mathewson, K., Branch, B., Winters, T., Verhoeven, B., and Elfving, J. (2020). Rosetta code: Improv in any language. In *Proceedings of the 11th International Conference on Computational Creativity*, pages 115–122. Association for Computational Creativity.

[96] Mishra, S. and Srivastava, S. (2021). Web development frameworks and its performance analysis—a review. *Smart Computing*, pages 337–343.

[97] Mishra, S. K., Puthal, D., Sahoo, B., Jayaraman, P. P., Jun, S., Zomaya, A. Y., and Ranjan, R. (2018). Energy-efficient vm-placement in cloud data center. *Sustainable computing: informatics and systems*, 20:48–55.

[98] Modzelewski, K. (2020). Pyston v2: 20% faster Python.

[99] Mukherjee, T., Banerjee, A., Varsamopoulos, G., Gupta, S. K., and Rungta, S. (2009). Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. *Computer Networks*, 53(17):2888–2904.

[100] Murri, R. (2013). Performance of Python runtimes on a non-numeric scientific code. page 6.

[101] Mytkowicz, T., Diwan, A., Hauswirth, M., and Sweeney, P. F. (2009). Producing wrong data without doing anything obviously wrong! *ACM Sigplan Notices*, 44(3):265–276.

[102] Nanz, S. and Furia, C. A. (2015). A comparative study of programming languages in rosetta code. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 778–788. IEEE.

[103] Noureddine, A. (2022). Powerjoular and joularjx: Multi-platform software power monitoring tools. In *18th International Conference on Intelligent Environments*.

[104] Noureddine, A., Islam, S., and Bashroush, R. (2016). Jolinar: analysing the energy footprint of software applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 445–448.

[105] Noureddine, A., Rouvoy, R., and Seinturier, L. (2015). Monitoring energy hotspots in software. *Automated Software Engineering*, 22(3):291–332.

[106] Oi, H. (2011). Power-performance analysis of jvm implementations. In *ICIMU 2011: Proceedings of the 5th international Conference on Information Technology & Multimedia*, pages 1–7. IEEE.

[Pang et al.] Pang, A., Anslow, C., and Noble, J. What Programming Languages Do Developers Use? a Theory of Static vs Dynamic Language Choice. page 9.

[Pankiv] Pankiv, A. Concurrent benchmark system for web-frameworks on Python. page 26.

[109] Pankiv, A. (2019). *Concurrent benchmark system for web-frameworks on Python*. PhD thesis, Ukrainian Catholic University.

[110] Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060):1226–1227.

[111] Pereira, R., Carcao, T., Couto, M., Cunha, J., Fernandes, J. P., and Saraiva, J. (2017a). Helping Programmers Improve the Energy Efficiency of Source Code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 238–240, Buenos Aires, Argentina. IEEE. ## main idea
This paper presents a technique to spot the energy leaks inside a program using a spectrum technique called SPELL (Spectrum-based Energy Leak Localization)
How it works
We have a matrix (n*m) where n is the number of the tests and m is the number of components (like classes, methods, packages, etc.) after this we calculate the oracle (a vector that says which component is responsible for what)
Contributions
This paper helps developers to spot the red areas in their code and optimize the energy consumption. As an example it helped to reduce the energy consumption of a java application 50% faster and with 18% more efficiency.

[112] Pereira, R., Carção, T., Couto, M., Cunha, J., Fernandes, J. P., and Saraiva, J. (2017b). Helping programmers improve the energy efficiency of source code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 238–240. IEEE.

[113] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., and Saraiva, J. (2017c). Energy efficiency across programming languages: how do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, pages 256–267.

[114] Philippot, O., Anglade, A., and Leboucq, T. (2014). Characterization of the energy consumption of websites: Impact of website implementation on resource consumption. In *ICT for Sustainability 2014 (ICT4S-14)*, pages 171–178. Atlantis Press.

[115] Pinto, G., Liu, K., Castor, F., and Liu, Y. D. (2016). A comprehensive study on the energy efficiency of java's thread-safe collections. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 20–31. IEEE.

[116] Redondo, J. M. and Ortin, F. (2015). A Comprehensive Evaluation of Common Python Implementations. *IEEE Software*, 32(4):76–84. benchmarks link http://www.reflection.uniovi.es/python.

[117] Ribic, H. and Liu, Y. D. (2016). Aequitas: Coordinated energy management across parallel applications. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–12.

[118] Rodriguez, A. (2017). Reducing energy consumption of resource-intensive scientific mobile applications via code refactoring. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 475–476. IEEE.

[119] Santos, E. A., McLean, C., Solinas, C., and Hindle, A. (2018). How does docker affect energy consumption? evaluating workloads in and out of docker containers. *Journal of Systems and Software*, 146:14–25.

[120] Simakov, N. A., Innus, M. D., Jones, M. D., White, J. P., Gallo, S. M., DeLeon, R. L., and FOPTurlani, T. R. (2018). Effect of meltdown and spectre patches on the performance of HPC applications. *CoRR*, abs/1801.04329.

[121] Sonnenwald, D. H., Whitton, M. C., and Maglaughlin, K. L. (2003). Evaluating a scientific collaboratory: Results of a controlled experiment. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(2):150–176.

[122] Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

[123] Tschanz, J., Kao, J., Narendra, S., Nair, R., Antoniadis, D., Chandrakasan, A., and De, V. (2002). Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *IEEE Journal of Solid-State Circuits*, 37(11).

[124] van der Kouwe, E., Andriesse, D., Bos, H., Giuffrida, C., and Heiser, G. (2018). Benchmarking Crimes: An Emerging Threat in Systems Security. *arXiv:1801.02381 [cs]*.

[125] van der Kouwe, E., Andriesse, D., Bos, H., Giuffrida, C., and Heiser, G. (2018). Benchmarking Crimes: An Emerging Threat in Systems Security. *CoRR*, abs/1801.02381.

[126] Varsamopoulos, G., Banerjee, A., and Gupta, S. K. S. (2009). Energy Efficiency of Thermal-Aware Job Scheduling Algorithms under Various Cooling Models. In *Contemporary Computing*, volume 40. Springer.

[127] Vasan, A., Sivasubramaniam, A., Shimpi, V., Sivabalan, T., and Subbiah, R. (2010). Worth their watts? - an empirical study of datacenter servers.

[128] von Kistowski, J., Block, H., Beckett, J., Spradling, C., Lange, K.-D., and Kounev, S. (2016). Variations in cpu power consumption. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 147–158.

[129] Wang, X., Zhao, H., and Zhu, J. (1993). Grpc: A communication cooperation mechanism in distributed systems. *ACM SIGOPS Operating Systems Review*, 27(3):75–86.

[130] Wang, Y., Nörtershäuser, D., Le Masson, S., and Menaud, J.-M. (2018a). Potential effects on server power metering and modeling. *Wireless Networks*, pages 1–8.

[131] Wang, Y., Nörtershäuser, D., Le Masson, S., and Menaud, J.-M. (2018b). Potential effects on server power metering and modeling. *Wireless Networks*.

[Wang et al.] Wang, Y., Nörtershäuser, D., Masson, S. L., and Menaud, J.-M. Experimental Characterization of Variation in Power Consumption for Processors of Different generations. page 10.

[133] Yet, A. W. F. (2016). Cross-language compiler benchmarking.