



University of Gloucestershire
School of Computing and Engineering
BSc in Computing

Green Lantern: A Novel Technique for Carbon-Aware DVFS

Kyle Dormer
S1802423
Supervisor: Dr. Thiago Viana

May 19th, 2023

Abstract

Climate change is one of humanity's biggest challenges in the modern day with potentially devastating consequences for life as we know it. Research in green computing has grown year by year as efforts to curtail the rise in global temperature have grown more expansive, with the aim of reducing the electricity consumed by computers and therefore reduce the carbon footprint. Existing literature demonstrates a variety of techniques that achieve reductions in electricity usage at the cost of performance, creating barriers to adoption of green computing techniques alongside other factors. Based on a review of existing solutions, this dissertation proposes and develops a technique that focuses not only on reducing the total amount of electricity consumed but considers the carbon intensity of the type of electricity consumed. Green Lantern is a novel DVFS technique that brings research in carbon-aware software and CPU DVFS techniques together alongside a CPU governor-based strategy for maximum compatibility. Based on CPU-bound and memory-bound workload benchmarking, it was found that Green Lantern achieves up to a 23% decrease in energy-usage accompanied by up to a 16% drop in performance, making it comparable to existing solutions while being a more widely compatible, more accessible and more secure green computing technique to implement.

Keywords: *DVFS, green computing, green computing adoption, carbon-aware, climate-change, carbon intensity, IT carbon footprint.*

Dedicated to Christian, Shelley, Louie and Mia. My beloved family.

Acknowledgements

I would like to give special thanks to my family who have helped me through thick and thin to get to where I am today. I would like to thank Dr Thiago Viana for his support throughout not only the dissertation but throughout university as a whole. I would like to give thanks to God, thanks to Father Mark, and thanks to the Orthodox Parish of the Kazan Icon of the Mother of God in Cardiff for their prayers and support throughout this process.

Ethical Issues

1. I declare that this dissertation has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree or anywhere else. Except where states otherwise by citation and reference or acknowledgement, the work presented is entirely my own.
2. I confirm that all the Tables and Figures in this dissertation are my works or a regeneration from other people's work with citation.
3. I confirm that all the citations in the dissertation have been provided in the bibliography and they are all accessible. In case the university wants to cross examine the citations, I can provide the content for the references which are not accessible.
4. I confirm that all the tools, software and datasets have been used in this dissertation followed the terms and conditions in their license agreement and university REC code of conduct.
5. I confirm that in this dissertation no human/animal study has been conducted.

Contents

1	Introduction	10
1.1	Overview	10
1.2	Problem Statement	10
1.3	Research Questions	11
1.4	Research Objectives	11
1.5	Scope	11
1.6	Conclusion	11
2	Literature Review	12
2.1	The Effectiveness of Green Computing	12
2.2	The Importance of Technique Accessibility	13
2.3	Carbon-Aware Software	15
2.4	Dynamic Voltage and Frequency Scaling	18
2.5	Research Gap	25
3	Research Methodology	28
3.1	Objective 1	28
3.2	Objective 2	30
3.3	Objective 3	31
3.4	Objective 4	33
4	Implementation	34
4.1	Requirements	34
4.2	User Interface	36
4.3	Algorithm	37
4.4	DVFS	38
4.5	Carbon-Awareness	39
4.6	Configuration	40
5	Results & Analysis	42
5.1	Experiment Design	42

5.1.1	Hypotheses	43
5.2	Baseline	43
5.2.1	Intel	43
5.2.2	AMD	43
5.3	Green Lantern	44
5.3.1	Intel	44
5.3.2	AMD	44
5.4	Accessibility	45
5.4.1	CPU-MISER	45
5.4.2	eDVFS	45
5.4.3	GG	46
5.4.4	FoREST	46
5.4.5	Green Lantern	46
5.5	Analysis	46
5.5.1	Memory-Bound Workloads	46
5.5.2	CPU-Bound Workloads	51
5.5.3	Accessibility	53
6	Conclusion	55
6.1	Strengths	55
6.2	Limitations	56
6.3	Future Research	56
	Appendices	62
A	Source Code	62
A.1	main.py	62
A.2	glutils.py	63
A.3	config.toml	72
A.4	Calculix Benchmark	72
A.5	Libquantum Benchmark	73

List of Tables

2.1	Barriers to adoption of green computing techniques.	15
2.2	Abbreviation table for research gap table.	25
2.3	Systematic literature review evaluating the validity of a software-based green computing approach.	26
2.4	Systematic literature review of carbon-aware software and DVFS techniques. . .	27
3.1	Libraries and APIs utilised in the implementation of Green Lantern.	31
3.2	AMD Ryzen desktop specification used in benchmarking Green Lantern.	32
3.3	Intel Haswell laptop specification used in benchmarking Green Lantern.	33
3.4	Framework for quantifying accessibility of green computing techniques.	33
4.1	Breakdown of fuel usage of North West England on the 18th May 2023. Source: NationalGrid (2023).	35
4.2	Breakdown of fuel usage of South Scotland on the 18th May 2023. Source: NationalGrid (2023).	35
4.3	Green Lantern implementation requirements.	36
4.4	Table containing a description of the Linux CPU governors used for Green Lantern.	38
4.5	Configuration options available to the end-user.	40
5.1	Green Lantern configuration options used in benchmarking.	42
5.2	Baseline libquantum benchmark results on Intel.	43
5.3	Baseline calculix benchmark results on Intel.	43
5.4	Baseline libquantum benchmark results on AMD.	43
5.5	Baseline calculix benchmark results on AMD.	44
5.6	Green Lantern libquantum benchmark results on Intel.	44
5.7	Green Lantern calculix benchmark results on Intel.	44
5.8	Green Lantern libquantum benchmark results on AMD.	44
5.9	Green Lantern calculix benchmark results on AMD.	45
5.10	Table summarising the accessibility of CPU-MISER.	45
5.11	Table summarising the accessibility of eDVFS.	45
5.12	Table summarising the accessibility of GG.	46
5.13	Table summarising the accessibility of FoREST.	46

5.14	Table summarising the accessibility of Green Lantern.	46
5.15	T-statistics and P-values for the memory-bound workload benchmark.	47
5.16	T-statistics and P-values for the CPU-bound workload benchmark.	52

List of Figures

2.1	An overview of the CPU-MISER algorithm.	19
2.2	Overview of the eDVFS algorithm.	21
2.3	Overview of the GG algorithm.	23
2.4	Overview of the FoREST algorithm.	24
3.1	An overview of the methodology utilised in this dissertation.	28
3.2	High-level overview of the benchmarking and evaluation strategy of Green Lantern.	32
4.1	Screenshot of command used to start Green Lantern.	36
4.2	Screenshot of Green Lantern in operation.	37
4.3	Pseudocode representation of the Green Lantern algorithm.	37
4.4	Flowchart representation of the Green Lantern algorithm.	38
4.5	Example API response for the regional carbon intensity API. Endpoint: api. carbonintensity.org.uk/intensity	39
4.6	Example API response for the national carbon intensity API. Endpoint: api. carbonintensity.org.uk/regional/postcode/GL52	39
4.7	Code used to parse the JSON response from carbon intensity API and adjust governor.	40
4.8	Illustrative example of Green Lantern’s configuration file.	41
5.1	Comparison of energy-savings on memory-bound workloads.	48
5.2	Comparison of performance-loss on memory-bound workloads.	49
5.3	Comparison of mean energy-usage on memory-bound workloads.	50
5.4	Comparison of mean execution time on memory-bound workloads.	51
5.5	Comparison of mean energy-usage on CPU-bound workloads.	52
5.6	Comparison of mean execution time on CPU-bound workloads.	53
5.7	Comparison of accessibility between techniques.	54

Chapter 1

Introduction

1.1 Overview

Climate change is one of the most significant problems the world currently faces. There is between 99% and 100% consensus among research scientists that anthropogenic climate change is real and that humanity is running out of time to counteract it (Powell, 2017; Lynas, Houlton, and Perry, 2021); failure to counteract climate change could have catastrophic effects for the environment (Barnes et al., 2019; Priestley, Heine, and Milfont, 2021), wildlife (Malhi et al., 2020; Mayhew, Jenkins, and Benton, 2008) and humans (Rocque et al., 2021; Hoffmann, Dimitrova, Muttarak, Crespo Cuaresma, and Peisker, 2020). As a result, the World Health Organisation (WHO) has estimated that an annual 250,000 deaths will be caused due to climate change from 2030 to 2050 (Hales, Kovats, Lloyd, Campbell-Lendrum, and Salud, 2014), a figure estimated to increase to 500,000 annually after 2050 (Springmann et al., 2016). The largest contributor to climate change is electricity production, specifically the release of greenhouse gases due to burning fossil fuels (IPCC, 2022). In 2020, ICT accounted for roughly 7% of global electricity usage (Andrae, 2020) and is estimated to increase to 14% by 2040 (Belkhir and Elmeligi, 2018). It is therefore a significant research problem to decrease the electricity usage and carbon footprint of ICT devices. There is an abundance of research focusing on increasing hardware and software efficiency in cloud computing data-centres (Bharany et al., 2022) however a significant problem faced in the literature is attempting to balance the need for 99.9% uptime with decreasing carbon footprint however this problem could be potentially alleviated by making servers carbon-aware (Anderson, Belay, Chowdhury, Cidon, and Zhang, 2022). This dissertation presents a novel technique that circumvents this dichotomy by integrating carbon-awareness alongside DVFS.

1.2 Problem Statement

How can a computer’s performance be scaled according to its locality’s real-time carbon intensity to decrease overall carbon footprint while not compromising on uptime and performance?

1.3 Research Questions

1. What are the current solutions to reducing carbon footprint by optimising software rather than hardware?
2. How accessible are these solutions to SMEs and consumers?
3. How significantly could electricity be reduced by limiting a computer's performance. How do the results of the proposed solution compare to other solutions?

1.4 Research Objectives

1. To discover and evaluate current solutions aiming to decrease carbon footprint purely through the optimisation of software.
2. To develop a technique to limit a computer's performance based on its geographical area's current carbon intensity.
3. To empirically demonstrate, evaluate and ensure the validity of the reductions in electricity usage in comparison to other solutions.
4. To assess that the developed technique is accessible to SMEs and consumers relative to other techniques.

1.5 Scope

The focus of this dissertation is on utilising a software solution to scale computer performance proportionally with the carbon intensity of its local geographical area. Computer is defined, for the purposes of this dissertation, as any bare-metal server or consumer device such as desktops and laptops. The scope of the technique's evaluation focuses on scaling alongside emission intensity (carbon intensity) measured in this dissertation as grams of CO₂ per megajoule of electricity produced; and on decreasing electricity usage measured as megajoules. Furthermore, this study pertains to the United Kingdom's energy grid though its findings may be applied elsewhere.

1.6 Conclusion

Energy optimisation techniques have been demonstrated to significantly decrease a computer's electricity usage when implemented correctly (Sriram, 2022). However, many of these techniques are not applicable or accessible to SMEs and consumers. This dissertation presents a novel technique and application for scaling performance based on local carbon intensity and with the aim of making the technique accessible to as many people as possible.

Chapter 2

Literature Review

This literature review further expounds upon the problem brought up in chapter 1 by reviewing the relevant literature and examining the state-of-the-art. This review aims to: evaluate the applicability of green computing to reduce carbon emissions and demonstrate why it is needed, evaluate current software-based techniques used for decreasing the carbon footprint of computers, identify the most effective techniques, assess said techniques for their effectiveness and accessibility and to highlight the gap in the current literature upon which this dissertation is based. Specifically, this literature review will discuss the effectiveness of green computing as an approach to reduce carbon footprint, the importance of technique accessibility and barriers to adoption affecting green computing techniques, carbon-aware software and various DVFS techniques.

2.1 The Effectiveness of Green Computing

Kurp (2008) was one of the first prominent academic reviews exploring the need for green computing considering both contemporary and future techniques, and outlines that according to the US Department of Energy (DOE) the predicted energy use by data-centres by 2011 was 100 billion kilowatt-hours (kWh), costing \$7.4 billion, up from 61 billion kWh, at a cost of \$4.5 billion in 2006 if there were no improvements in efficiency (Brown et al., 2007). On the contrary to this prediction, a report endorsed by the DOE Federal Energy Management Program found that by 2014 US data centres were consuming approximately 70 billion kWh (Shehabi et al., 2016). There were multiple reasons for this significant slowing in growth. Firstly, the 2008 recession caused a drop in demand for electricity and a reduction in new data-centres being established (Shehabi et al., 2016), however a more significant reason was that there were improvements in virtualisation technology and subsequent widespread adoption of the technology (Kooimey et al., 2011). These improvements meant being able to host multiple virtualised servers on one powerful bare-metal server, which brought significant electricity savings (Ruth, 2011).

The downside of consolidating more virtualised servers onto one bare-metal server means that,

in theory *more* electricity would be used as this means that the servers in question are running at higher CPU utilisation on average. This point was put forth by Niles and Donovan (2008) in a contemporary whitepaper published by ComTec Power, a business focused on data-centre design (Limited, 2023). However, Masanet, Brown, Shehabi, Koomey, and Nordman (2011) in an attempt to model US data-centre electricity usage more accurately than previous attempts delved in-depth into the pros and cons of taking a virtualisation approach for electricity saving and demonstrated empirically that any increases in electricity usage on a per-server basis are outweighed by two factors: firstly, virtualisation meaning that less physical servers are needed, especially legacy servers; secondly, utilisation of dynamic voltage and frequency scaling techniques can help to decrease the power usage and temperature of the servers while minimising loss in performance.

This demonstrates that development of green computing techniques contributed significantly towards saving approximately 30 billion kWh of electricity. According to figures published by the US Energy Information Administration (EIA) the amount of CO₂ emitted per kWh is 0.855 pounds (EIA, 2021a). This means that an estimate of the amount of carbon emissions prevented partially by developments in green computing can be found by multiplying this figure by 30 billion, yielding 25.65 billion pounds of CO₂, or 11.63 million metric tons. However, it should be noted that this estimation relies upon the veracity of the 2007 DOE report that modelled an increase to 100 billion kWh and upon the EIA’s 2021 figures. Despite this, these caveats do not skew the estimation significantly. This is because, according to the EIA’s figures on US energy usage per source, since 2007 the US has increased its usage of renewable energy from 6.523493 quadrillion British Thermal Unit (Btu) to 12.163188 quadrillion Btu in 2021. At the same time, consumption of fossil fuels decreased from 85.805032 quadrillion Btu in 2007 to 77.430489 quadrillion Btu in 2021 (EIA, 2021b). This means that the estimate given is likely to be *lower* than actual CO₂ savings by 2011.

2.2 The Importance of Technique Accessibility

Thus far, it has been established that developments in green computing are needed, and that they can be very effective in practice. However, a rarely discussed issue in the field of green computing research is that of accessibility¹. There are various barriers to adoption that diminish the accessibility of green computing techniques, both for top-down reasons and bottom-up reasons. Regardless of the efficacy of a particular green computing technique, no decrease in carbon footprint is possible if the technique is not adopted. For this reason, it is important to the development of the technique proposed in this dissertation to identify barriers to adoption and to design a technique that does not have them.

Ahmad, Khan, Khan, Khan, and Ilyas (2021), in a review of 68 extant papers, identified the 9 most significant challenges preventing more widespread adoption of green computing tech-

¹Defined as how easy it is to access and implement a given green computing technique.

niques. These can be found in table 2.1, alongside accessibility issues identified by other literature reviewed. It is clear from these challenges that most of them are caused by current green computing techniques decreasing performance and scalability, especially when dealing with volatile loads. However, the authors of this paper state that it focuses only upon the perspective of clients² as opposed to vendors³. For adoption, a technique needs to be adopted with the support of both clients *and* vendors. Therefore, it is also important to survey the literature to understand the challenges and potential barriers that vendors face in implementing a new green computing technique. Jayalath, Chathumali, Kothalawala, and Kuruwitaarachchi (2019), in a review of specific green computing implementations of different cloud computing vendors, found that various large data centre providers are utilising a variety of techniques with using renewable energy sources being the most frequent one. This review has two issues. Firstly, it only includes AWS, Microsoft, Google and IBM. These are very large corporations with the manpower and capital to implement green computing approaches such as building entirely new data-centres near renewable power sources. In the case of Microsoft, they are even experimenting with using underwater data-centres utilising submarine technology. For most organisations this approach is simply not financially or logistically feasible, and going into the future with the rise of edge computing⁴, it won't be a valid solution even for the largest of organisations (Satyanarayanan, 2017). Secondly, this review makes use of only one source for most of its findings: a 2017 Greenpeace report on data-centre sustainability. For widespread adoption the technique developed in this dissertation needs to be accessible not only to organisations on the scale of AWS and Microsoft but to smaller cloud-computing providers which according to Synergy market research group made up 37% of market-share in Q3 of 2021 (Group, 2021).

²Defined as the end-user.

³Defined as the provider of a service, such as AWS.

⁴Involving the placement of 'mini-clouds' within urban areas.

Barrier to Adoption	Author(s)
Lack of quality of service	Ahmad, Khan, Khan, Khan, and Ilyas (2021)
Lack of dynamic response	Ahmad, Khan, Khan, Khan, and Ilyas (2021)
Lack of services to address client's specific requirements	Ahmad, Khan, Khan, Khan, and Ilyas (2021)
Lack of awareness of green computing	Ahmed (2018)
High cost	Ahmad, Khan, Khan, Khan, and Ilyas (2021), Jayalath, Chathumali, Kothalawala, and Kuruwitaarachchi (2019), Ahmed (2018), Dalvi-Esfahani and Nilashi (2022)
SLA violation	Ahmad, Khan, Khan, Khan, and Ilyas (2021)
Lack of efficient provision of resources and services	Ahmad, Khan, Khan, Khan, and Ilyas (2021)
Lack of green and sustainable infrastructures for the provision of reliable and cheaper services	Ahmad, Khan, Khan, Khan, and Ilyas (2021)
Lack of cybersecurity	Ahmad, Khan, Khan, Khan, and Ilyas (2021)
Complexity	Ahmad, Khan, Khan, Khan, and Ilyas (2021), Ahmed (2018)

Table 2.1: Barriers to adoption of green computing techniques.

Furthermore, it's important to consider barriers to adoption not only to cloud computing providers but also to other organisations that operate servers, who face different barriers than those of cloud computing vendors such as Microsoft and Amazon. One type of organisation in this category are universities. Ahmed (2018), surveyed 188 universities in the Gulf and found that there are various factors that affect the likelihood of adopting green computing technology. Particularly significant among these factors were awareness of green computing, relative advantage, management support and adequate resources. Similarly, Dalvi-Esfahani and Nilashi (2022) found that adoption and switching cost are significant barriers in the adoption of green computing. With this in mind, this dissertation sets out to develop a technique that is not only effective but also accessible both in terms of cost and in terms of integration.

2.3 Carbon-Aware Software

Carbon-aware software, as defined by Anderson et al. (2022), is software with clearly visible and clearly defined providence for the electricity that it consumes during runtime. Furthermore, carbon-aware software allows for fine-grained control over the type of electricity it consumes and the carbon footprint it produces. Anderson et al. (2022) note both the lack of polished

carbon-aware approaches and the need for more research to be done towards this end and thus present a research agenda aimed at data-centres specifically. According to them, minimising software bloat, implementation of new interchangeable DRAM memory, interchangeable computing utilising GPUs rather than CPUs for specific workloads and energy-aware scheduling are important points of focus. However, there are several issues with this assessment due to the fact that half of the suggested approaches are hardware-based, despite noting in their paper that the slowing of Moore’s Law increases the need for green computing⁵. According to newer research into the exact carbon footprint of both corporate and personal computing by Gupta et al. (2021), it is the manufacturing of computers that makes up the largest proportion of its carbon footprint, even despite increasing rates of green technology being utilised in the manufacturing and transport industries. Indeed, one of the suggested approaches according to their findings is to focus on the development of green computing software. The longer that hardware can be utilised, particularly alongside green computing software, the lower its carbon footprint will be.

On the other hand, there are detractors of the software approach in favour of a hardware focused approach towards green computing. Kazandjieva, Heller, Gnawali, Hofer, and Kozyrakis (2011) assert that software approaches yield results that are “not worth the trouble”. The premise of their paper is to take a simple approach to a topic that they deem to have been overcomplicated in the research; they state that instead of using state-of-the-art power-saving approaches, desktops should be replaced with laptops. There are several reasons that this argument misses the mark. Firstly, the authors admit in their paper that the replacement of laptops may save power but also increases costs alongside being less capable of handling intensive workloads. As discussed in section 2.2, cost is a significant barrier to the adoption of green computing and should not be overlooked. Secondly, although the authors include analysis on the electricity usage of whole enterprise buildings, they do not address the carbon footprint involved with replacing one type of device with another, including the carbon footprint of the increased manufacturing demand. Finally, the authors’ assertion that software techniques may not be worth implementing in comparison to developing more efficient hardware can be disproven by looking into results achieved by new carbon-aware software techniques, as follows below.

Zhao and Zhou (2022) developed a carbon-aware algorithm focused on distributed cloud data centres that allocates compute tasks based on which data-centre uses the cleanest energy. The algorithm utilised data supplied by various local agencies and the results demonstrated a significant 73% renewable energy utilisation. Though this is significant, the technique relies on US-specific data and is thus not applicable to the UK, as well as the algorithm being subpar in terms of accessibility due to the fact that there is no open-source implementation and it is specific to organisations large enough to have multiple data-centres. Furthermore, this solution focuses purely on allocation of work to reduce carbon footprint as well as being limited

⁵The idea that the number of transistors per integrated circuit chip doubles roughly every 2 years (Mack, 2011).

to virtual machines, further reducing accessibility. Another technique, developed by Alvi, Sahar, Bangash, and Beg (2017), remedies some of these drawbacks but comes with its own. It provides a technique in the form of an Android Studio plugin to parse source code in order to estimate the energy consumption that the resulting program will consume, with the aim of facilitating the development of carbon-aware software. In contrast to the work of Zhao and Zhou (2022), this solution improves accessibility by being integrated into an IDE plugin, thus decreasing barriers to adoption among organisations and individual developers. Alongside this, the solution focuses not on the allocation of work but the carbon footprint of the work itself. However, rather than measuring actual energy consumption of the program, it models it based on source code. In theory, this concept could provide an easy and accessible way for developers to adopt green computing. In practice however, the technique only reports a 53.13% success rate in detecting energy change between two iterations of the same program. Therefore this approach requires further refinement and shall not be utilised for Green Lantern.

Rogers and Parson (2019) developed yet another approach towards carbon-aware software, an approach that is applicable to the UK. The technique utilises metrics obtained from the UK National Grid, such as renewable vs non-renewable fuel generation statistics, and calculates the current carbon intensity of the UK electricity grid. This is a strong approach towards carbon-aware software for several reasons. Firstly, it is highly accessible for consumers in the form of a smartphone app. Secondly, it reports concrete statistics in contrast to Alvi et al. (2017). However, the technique is still lacking in terms of accessibility. Although it is accessible to consumers, it is not accessible to organisations and developers. There is no open-source API or source code that can be used to integrate the technique in order to effectively create carbon-aware software localised for the UK.

Fortunately, the work of Rogers and Parson (2019) was expanded upon in Alasdair, Ruff, Kelloway, MacMillan, and Rogers (2020) ameliorating some of these concerns. This technique expands upon the national grid calculations established previously and utilises a variety of supervised machine learning models in unison via ensemble learning to forecast the carbon intensity over a 96 hour period, updated every 30 minutes. Even more pertinent to this dissertation is the fact that a free and public REST API has been created in order to better facilitate the development of carbon-aware software, thus solving the concerns regarding accessibility. Finally, this technique is not only more accurate than previous techniques but it also allows for forecasting of carbon intensity at the regional level, as opposed to solely the national level. This makes it especially useful for this dissertation as it supplies a simple yet robust method of applying different energy optimisations based on the local carbon intensity. The primary drawback of this technique is that it doesn't directly decrease carbon footprint, it only facilitates new techniques that achieve this. In order to decrease carbon intensity, a different category of green computing known as Dynamic Voltage and Frequency Scaling (DVFS) can be utilised, as discussed next.

2.4 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) is a power management technique that upscales and downscales the frequency and voltage of a CPU with the aim of reducing the amount of electricity consumed. A common issue with DVFS techniques is that there is a trade-off between performance and electricity consumption. An effective DVFS technique is one that achieves significantly more energy-savings without significantly decreasing performance.

Ge, Feng, Feng, and Cameron (2007) developed a DVFS technique known as CPU-MISER that allows for system-wide, application-independent DVFS as well as constraining performance loss within user-defined limits. The basis of this particular technique is to exploit phases of inefficiency within the CPU, specifically memory accesses, I/O accesses and system idle, in order to reap power efficiency benefits. The algorithm of CPU-MISER can be seen in figure 2.1. It relies upon the observation that while CPU power decreases significantly as CPU frequency decreases, the actual performance of the CPU decreases significantly slower despite the lower frequency. Armed with this knowledge, Ge et al. (2007) proposed a prediction model that splits a given workload into intervals representing execution phases of the CPU. Subsequently, the amount of time taken to complete a workload at a given CPU frequency is calculated per-interval. While one interval is being executed, CPU-MISER predicts the workload in the next interval and scales the CPU voltage and frequency accordingly, in order to both maximise performance and energy-savings for that particular workload. This approach, when tested with a memory-intensive workload, was able to achieve up to 20% energy savings with a mere 4% performance loss. On the other hand, when applied to CPU-intensive workloads the authors conceded that there was little to no energy savings yielded and didn't give precise figures. Alongside this gap in the reporting of CPU-MISER's efficacy with unfavourable workloads there are several other issues with it compared to other solutions.

Firstly, CPU-MISER lacks accessibility. It is designed specifically around High-Performance Computing (HPC) clusters, whereas Green Lantern aims to be system-agnostic as far as it is possible. Furthermore, the authors have not released the source code for CPU-MISER nor do they present pseudocode to aid in the implementation of their technique by third-parties, creating barriers to adoption and thus diminishing the chance of the technique having an actual impact on carbon footprint. Secondly, the technique is built specifically for the AMD Athlon and Opteron CPUs and tested only on these CPUs. Not only does this decrease accessibility, it raises questions regarding the efficacy of the technique on other CPU architectures. Rather, a truly accessible DVFS technique should aim for wider accessibility even at the cost of producing inferior energy savings. It should also be evaluated for its performance on a variety of CPUs, with a focus on those that are widely used in industry.

Kim, Choi, Eom, Yeom, and Byun (2012) developed another DVFS technique, named "eDVFS" that addresses some of these issues while also achieving significant energy savings. The approach taken in this instance acknowledges that energy savings can be achieved at lower

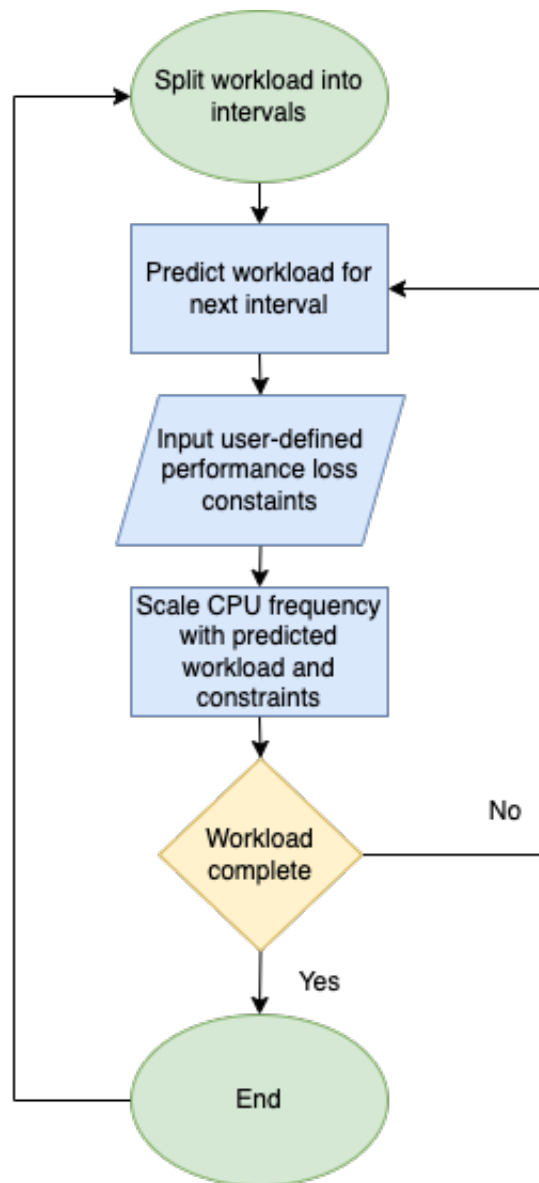


Figure 2.1: An overview of the CPU-MISER algorithm.

CPU frequencies, similar to Ge et al. (2007), but also presents the argument that the electricity consumed for a given workload can alternatively be reduced by utilising a significantly higher frequency but for a shorter period of time, especially in the context of data-centres which utilise virtualised servers sharing physical resources. eDVFS is based around the premise that a higher degree of energy efficiency can be achieved by keeping shared physical resources, such as the memory bus, as highly utilised as possible in order to minimise idle time and thus ensure that all the electricity being consumed by the CPU is actively being used. On the other hand, if shared resources are overloaded, the CPU frequency should be lowered to save energy while other bottlenecks exist. An overview of this algorithm can be observed in figure 2.2. The technique was tested against the in-built Linux *ondemand* CPU governor on specifically Intel Xeon CPUs. The results demonstrated varying results depending on the type of workload. The authors provide specific benchmarks for their technique, demonstrating its efficacy but they do not provide benchmarks of the Linux *ondemand* governor to compare their technique to, they only provide a comparison to a variety of static CPU frequencies. The purpose of this governor is not to remain at a static frequency but rather to scale alongside the current workload (Pallipadi and Starikovskiy, 2006). This misconception in the author’s benchmarking doesn’t necessarily disprove the validity of their technique but calls into question their claims that it is a superior solution to the Linux *ondemand* governor, as the governor was artificially constrained to a specific frequency. Furthermore, while their evaluation is superior to Ge et al. (2007) as it includes a wider variety of realistic workload benchmarks, it is still only tested on one type of CPU. Finally, while the accessibility of this technique is superior to Ge et al. (2007) as it includes pseudocode it lacks published source code and is designed specifically for the context of virtualisation.

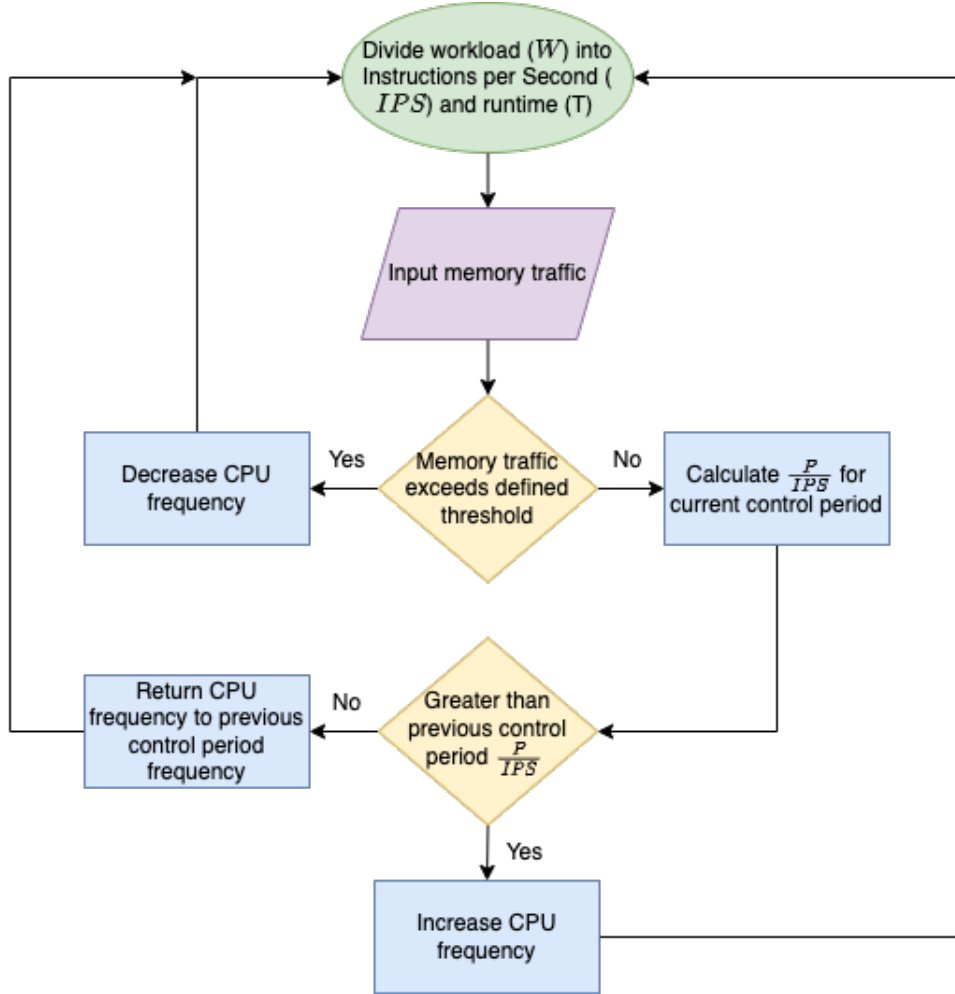


Figure 2.2: Overview of the eDVFS algorithm.

Comparatively, an approach taken by Spiliopoulos, Kaxiras, and Keramidas (2011), dubbed ‘*Green Governors*’ (GG) achieves up to 56% energy savings in memory-bound workloads, with 5% performance loss. GG was developed for multiple CPU architectures, the technique being compatible with and benchmarked on both AMD Phenom and Intel i7 processors. Furthermore, the authors also provide pseudocode to develop their algorithm, boosting accessibility even further. Similar to the other techniques discussed, GG exploits momentary idleness in the CPU while it is busy accessing other resources such as memory yet the frequency remains high, potentially wasting electricity depending on the workload. Where this technique diverges is that it takes the same concept as Ge et al. (2007) and provides better implementation through more in-depth monitoring and modelling of metrics obtained from the CPU. GG specifically targets long-latency off-chip memory operations by monitoring the number of Last-Level Cache (LLC) misses, similar to the other techniques discussed. The authors split workload into *miss-intervals*, which are defined in this instance as memory miss events that interrupt the ‘steady-state’ of a given workload’s execution. A given miss-interval lasts until the Instructions Per Cycle (IPC) reaches a steady state once more. The authors presented a crucial point to be

taken into consideration for the development of future DVFS techniques: in any given miss-interval, only the first miss in a cluster of misses scales with CPU frequency. Simply put, rather than multiplying the total number of stall-cycles by memory latency to obtain the predicted CPU frequency a more accurate prediction can be obtained by instead multiplying the number of *miss clusters* by memory latency. However, the author concedes that with current CPUs lacking the inbuilt hardware sensors to distinguish between overlapping and non-overlapping LLC misses this approach is not yet able to be implemented in production environments as only a rough approximation can be calculated. An overview of the author’s technique can be viewed below in figure 2.3. One of the primary innovations taken by the author in comparison to the previous techniques presented, and the reason for its superior CPU compatibility is that they built their technique into Linux kernel CPU governors. Despite the efficacy of this technique, it still falls short in terms of the degree of accessibility that Green Lantern hopes to achieve, because this approach requires users to manually patch their Linux kernel, which is a complex process and opens up a variety of security concerns, as kernel patches typically require extensive code review and testing(Jiang, Adams, and German, 2013).

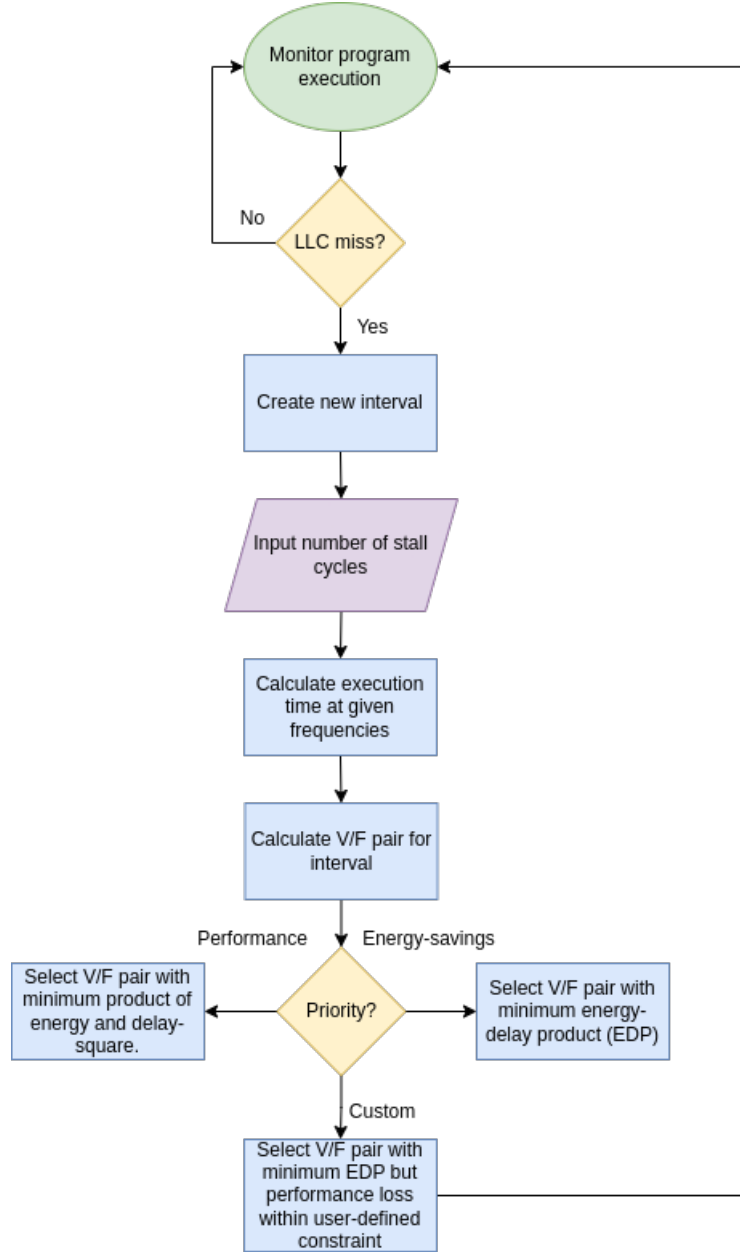


Figure 2.3: Overview of the GG algorithm.

Finally, yet another technique for DVFS was designed by Halimi et al. (2013), named FoREST. This technique, unlike Ge et al. (2007) and Spiliopoulos, Kaxiras, and Keramidas (2011) takes advantage of new hardware sensors being introduced into CPUs. Rather than using an algorithmic performance model it directly measures the effects of changing the CPU frequency in terms of energy usage. The results achieved by this technique are up to 39% CPU energy-savings compared to the Linux *ondemand* CPU governor with a slowdown of 5%, though this constraint can be changed by the user. This technique, rather than viewing a user-defined performance constraint as a target, uses hardware monitoring to explicitly enforce it. To achieve this, the technique is split into three phases: offline analysis, evaluation and execution. An offline anal-

ysis is completed prior to any DVFS being executed in order to determine the precise ratio of frequency to power consumption for the specific CPU the technique is being used with. This allows the technique to yield higher energy-savings than more generalist techniques such as Ge et al. (2007). It is precisely due to the addition of this phase that the user-defined performance-loss constraint can be enforced more effectively than other techniques. Next, the evaluation phase involves determining which frequency to use in order to achieve the best performance alongside maximal energy-savings. Power gains are computed for each frequency and then computed in unison across all cores of the CPU in order to ensure that a frequency beneficial for single-core energy-saving but not in multi-core processing isn't selected. Once this has been completed, the ratio that provides the highest possible energy-savings with the minimum loss to performance is selected. This marks the start of the execution phase, in which the selected frequency and voltage is actually implemented across the CPU.

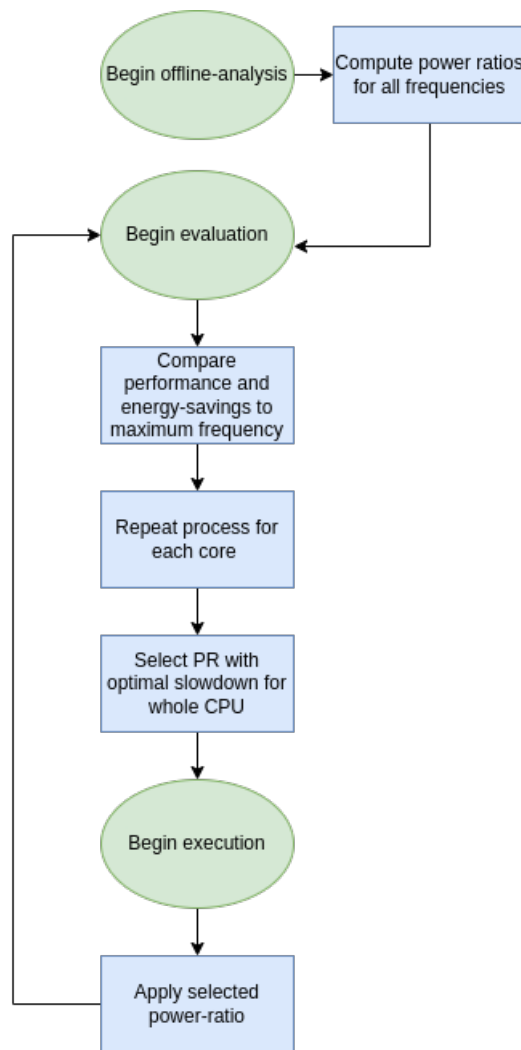


Figure 2.4: Overview of the FoREST algorithm.

2.5 Research Gap

The primary research gap is the combination of DVFS with carbon-aware software techniques. Rather than chasing energy-savings at all costs and thus having to inevitably face the issue of performance-loss, performance can instead be scaled with the local carbon intensity. Green Lantern is a novel proof-of-concept of this combination of techniques. This combination is advantageous to other existing solutions for several reasons, outlined below. Finally, it is also clear that there is a lack of research focusing on solutions designed specifically for the UK.

- Full CPU performance with no drawback, assuming clean energy is being used.
- Decreased carbon footprint.
- Simplified technique, both in design and implementation, allowing for increased adoption.
- Cross-CPU compatibility.

Below a table can be found two tables. Table 2.2 contains the key of abbreviations utilised in tables 2.3 and 2.4, tables demonstrating the gap in the literature.

Abbreviation	Value
ES	Energy-Savings
PL	Performance-Loss
OS	Open-Source
CA	Carbon-Aware
PC	Pseudocode
T	Type
EU	Energy Usage
D	Date
C	Cost
HS	Hardware or Software
CA	Concerns with Accessibility
COS	Concerns with Security
COC	Concerns with Cost
CCPU	Compatible CPUs
UK	UK Solution
TAO	Tested Against Other Techniques
Y	Yes
N	No
N/A	Not Applicable
UD	User-Defined
ABS	Absent
RS	Recommended Solution
CO ₂	Carbon-Dioxide Savings

Table 2.2: Abbreviation table for research gap table.

Author	EU	D	C	RS	ES	CO₂	T
Kurp (2008)	100B	2011	\$4.5B	N/A	N/A	N/A	Article
Shehabi et al. (2016)	70B	2014	N/A	Virtualisation	30B	N/A	Government
Niles and Donovan (2008)	70B	2014	\$240K	Hardware	76%	N/A	Whitepaper
Masanet, Brown, Shehabi, Koomey, and Nordman (2011)	80B	2008	N/A	DVFS	56B	40-80%	Article

Table 2.3: Systematic literature review evaluating the validity of a software-based green computing approach.

Author	ES	PL	PC	OS	UK	COC	CCPU	TAO	HS	CA	COS
Anderson, Belay, Chowdhury, Cidon, and Zhang (2022)	N/A	N/A	N	N/A	N	Y	N/A	N/A	HW/SW	Y	N
Gupta et al. (2021)	N/A	N/A	N	N/A	N	Y	N/A	N/A	SW	N	N
Kazandjieva, Heller, Gnawali, Hofer, and Kozyrakis (2011)	50%	ABS	N/A	N/A	N	Y	N/A	Y	HW	Y	N
Zhao and Zhou (2022)	73%	ABS	N	N	N	Y	N/A	N	SW	Y	N
Alvi, Sahar, Bangash, and Beg (2017)	ABS	ABS	N	N	N	N	ALL	N	SW	N	N
Rogers and Parson (2019)	N/A	N/A	N	N	Y	N	ALL	N	SW	Y	N
Alasdair, Ruff, Kelloway, MacMillan, and Rogers (2020)	N/A	N/A	N	N	Y	N	ALL	N	SW	N	N
Ge, Feng, Feng, and Cameron (2007)	20%	4%	N	Y	N	N	AMD	Y	SW	N	N
Kim, Choi, Eom, Yeom, and Byun (2012)	4-20%	UD	Y	N	N	N	Intel	N	SW	Y	N
Spiliopoulos, Kaxiras, and Keramidas (2011)	41-56%	10%	N	N	N	N	ALL	N	SW	Y	Y
Halimi et al. (2013)	15-39%	5%	N	Y	N	N	Intel	Y	SW	Y	N

Table 2.4: Systematic literature review of carbon-aware software and DVFS techniques.

Chapter 3

Research Methodology

This chapter contains the methodology utilised in this dissertation. It is divided into sections representing the research objectives set out in chapter 1. Firstly, an explanation is provided of the systematic literature review and the gap identified in the literature. Secondly, an overview of the proposed solution is set out with the steps required to implement it and the justification for each step, including a discussion regarding how the technique is designed in order to minimise barriers to adoption. Finally, metrics and benchmarks are outlined in order to test the efficacy of the technique against a baseline and in comparison to other techniques. An overview of the methodology can be observed in figure 3.1.

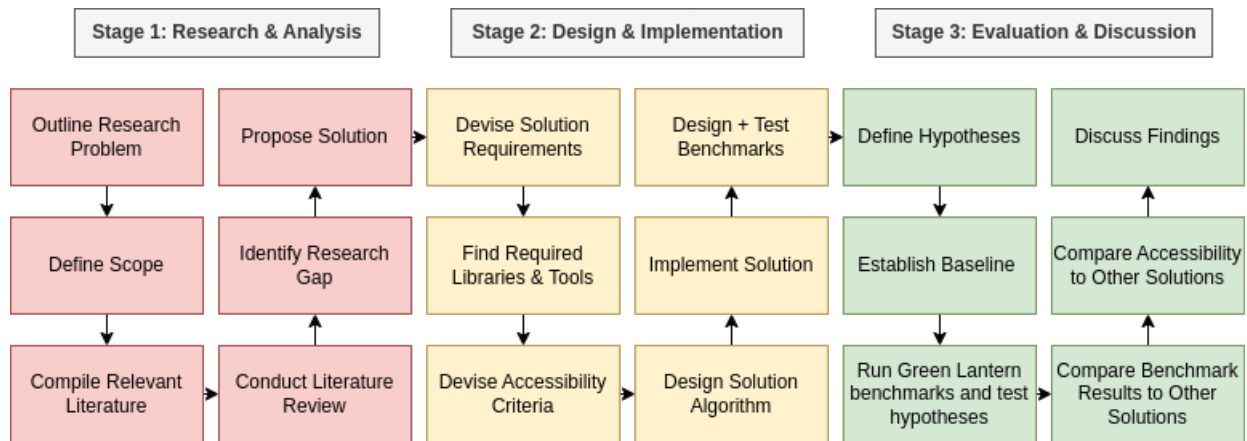


Figure 3.1: An overview of the methodology utilised in this dissertation.

3.1 Objective 1

To discover and evaluate current solutions aiming to decrease carbon footprint purely through the optimisation of software.

In order to achieve this objective a systematic literature review is going to be carried out on

relevant literature, focusing on the validity of green computing, current carbon-aware software solutions, accessibility issues creating barriers to adoption and various DVFS approaches. A list of keywords is to be created in order to compile relevant literature. Papers are to be collected from Google Scholar, IEEE Xplore, Science Direct, ACM Digital Library and Springer. These are reputable databases of peer-reviewed research, ensuring that only the best quality research is analysed. Alongside this, where required, sources can be collected from outside of academia, for example industry reports and power-usage statistics from government agencies. After a variety of papers are collected, further criteria needs to be created to include only the most relevant papers.

The keywords utilised for this search can be found below:

- Green computing
- Green computing accessibility
- Green computing adoption barriers
- Carbon-aware
- Energy-aware
- DVFS

The inclusion criteria for papers selected for the systematic literature review were the following:

- **IN1:** paper is concerned with green computing, whether it be a technique or review.
- **IN2:** paper is concerned with barriers to software adoption and accessibility.
- **IN3:** paper is concerned with the validity of hardware versus software solutions in green computing.
- **IN4:** paper presents a tangible DVFS technique with accompanying evaluation.

Furthermore, the exclusion criteria for papers selected for the systematic literature review were the following:

- **EX1:** paper written in a language other than English.
- **EX2:** paper published prior to 2005.
- **EX3:** paper is a bachelor's thesis or dissertation.
- **EX4:** technique presented is purely theoretical with no implementation and/or evaluation.

3.2 Objective 2

To develop a technique to limit a computer’s performance based on its geographical area’s current carbon intensity.

The aim of this objective is to design and implement a solution that addresses the identified gap in the literature. In order to do this the literature review is analysed and a set of actionable requirements are to be produced for the development of the technique, which can be found in chapter 4.

Although specialised DVFS techniques can achieve a very significant energy-savings to performance-loss ratio, such as Ge et al. (2007) and Kim et al. (2012) which are designed specifically for Intel Haswell and AMD Athlon CPUs, this approach creates barriers to adoption and reduces the potential net carbon footprint reduction that a technique could achieve, in terms of being utilised by as many organisations as possible. Thus, Green-Lantern uses in-built Linux CPU governors that exist but are rarely used. By default Linux distributions only utilise the *ondemand* and *powersave* governors. Other governors, such as *conservative* and *performance* are already well-implemented and well-tested, maximising the accessibility of Green Lantern and reducing barriers to adoption in terms of security-risks present in DVFS approaches such as patching the Linux kernel, the technique used by Spiliopoulos, Kaxiras, and Keramidas (2011).

Python is the programming language selected for the implementation of Green Lantern because it is well-supported on Linux and there are a wide range of pre-written libraries and frameworks that aid in the implementation. The libraries and APIs utilised for Green Lantern as well as their justification can be found in table 3.1.

In order to reliably obtain the local carbon intensity of a server, the UK National Grid’s Carbon Intensity API is to be used. The API is based on research completed by Alasdair et al. (2020). Supplied with real-time information from the national grid, the API utilises machine learning to forecast the carbon intensity¹ 96 hours in advance and updated every 30 minutes. Furthermore, it has the capability of giving the carbon intensity for specific regions in the UK, allowing a finer-grain of carbon-awareness than using the carbon intensity of England or of the UK as a whole. The API has a variety of endpoints for various purposes, such as statistics of fuel-type used in Britain in a specific time-frame. However, Green Lantern is only concerned with national carbon intensity and regional carbon intensity. The National Grid’s carbon intensity API was chosen over alternatives, such as Elexon’s API, because it utilises cutting-edge research to forecast carbon intensity and does not require an account, which would create more friction to adoption. On the other hand, Elexon only provides real-time information, requires an account and has inferior documentation.

To reduce the barriers to adoption of Green Lantern, as established in chapter 2, it is to be made configurable by end-users in order to fit their specific needs. For user configuration, ‘Tom’s

¹Measured in gCO₂/kWh.

Obvious Minimal Language’ (TOML) is used as it allows for easy and simple configuration that allows the end-user to focus on configuration rather than learning a markup language, especially when compared to other commonly used alternatives such as YAML² and JSON³.

Name	Description
<code>toml</code>	Provides support for easy parsing of toml configuration files.
<code>ipinfo</code>	Provides support for obtaining the user’s public IP address and ascertaining their approximate location in order to monitor carbon intensity.
<code>cpuinfo</code>	Provides support for ascertaining the user’s CPU brand and type to ensure compatibility prior to executing Green Lantern.
<code>cpufreq</code>	Provides support for monitoring and controlling the CPU frequency and governor on Linux systems.
<code>requests</code>	Provides support for sending HTTP GET requests to interface with the National Grid Carbon Intensity API.
<code>time</code>	Provides support for a delay in-between polling local carbon intensity.
<code>random</code>	Provides support for giving a random carbon intensity within an upper and lower limit, used for testing and benchmarking.

Table 3.1: Libraries and APIs utilised in the implementation of Green Lantern.

3.3 Objective 3

To empirically demonstrate, evaluate and ensure the validity of the reductions in electricity usage in comparison to other solutions.

In order to achieve this objective, several things are required. First of all, a means of measuring the electricity usage is required. In order to achieve this, PowerAPI’s `jouleit` tool is used. This uses “Running Average Power Limit” sensors, present on CPUs since 2013, in order to obtain the amount of electricity in joules that a specific program uses throughout its execution. Secondly, using `jouleit` to measure the electricity usage, several benchmarks need to be completed to test the efficacy of the technique both in CPU-bound workloads and in memory-bound workloads, which is the approach used by Ge et al. (2007), Spiliopoulos, Kaxiras, and Keramidas (2011), Kim et al. (2012) and Halimi et al. (2013). For CPU-bound benchmarking `calculix` is used to perform finite-element analysis, an approach utilised by Spiliopoulos, Kaxiras, and Keramidas (2011), and aims to simulate a realistic CPU-bound workload instead of using artificial CPU stress-tests such as `sysbench`. For memory-bound benchmarking `libquantum` is used to run quantum mechanics simulations. This benchmark uses Shor’s algorithm to find prime factors of a large integer, an approach also utilised by Spiliopoulos, Kaxiras, and Keramidas (2011) and Kim et al. (2012).

In order to minimise unknown factors potentially producing anomalous results, each benchmark is run 3 times and averaged using the mean of each result. Next, a baseline result is established

²‘Yet Another Markup Language’.

³‘JavaScript Object Notation’.

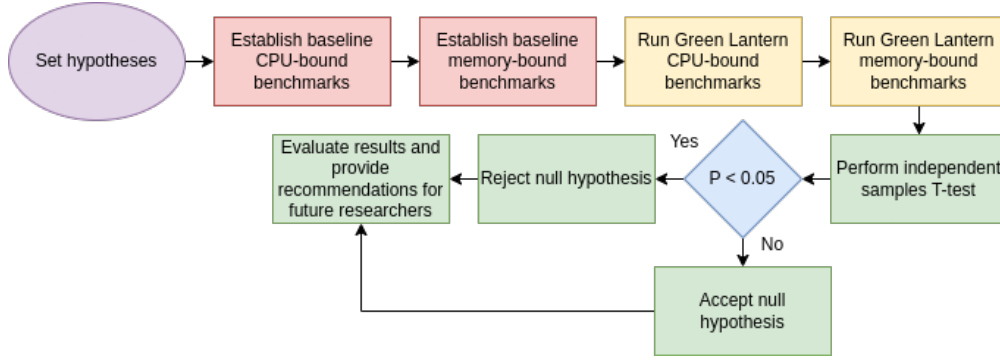


Figure 3.2: High-level overview of the benchmarking and evaluation strategy of Green Lantern.

in this fashion which is to run the benchmarks with the Linux *ondemand* governor, an approach taken by Spiliopoulos, Kaxiras, and Keramidas (2011), Kim et al. (2012), and Halimi et al. (2013). Once this baseline result has been established, this forms the basis of the null hypothesis. The results of benchmarking Green Lantern will then be used in an independent samples T-test to evaluate if the results are statistically significant and if the null hypothesis can thus be rejected. This type of T-test is chosen because each benchmark provides exactly two independent populations of data. Alongside testing the hypothesis, the results of Green Lantern are compared to the other DVFS solutions discussed in chapter 2.

Throughout the process of benchmarking all variables are kept the same in a controlled environment. Green Lantern is tested on both AMD and Intel CPUs specifically the Ryzen and Haswell architectures respectively in order to check if the efficacy of the technique differs between them. This benchmarking approach was taken by Spiliopoulos, Kaxiras, and Keramidas (2011) but lacking in Ge et al. (2007) and Kim et al. (2012). Furthermore, Green Lantern is benchmarked on two different categories of device: a laptop and desktop. The specific details of these devices can be found in tables 3.2 and 3.3.

A high-level overview of the benchmarking, evaluation and analysis process can be observed in figure 3.2.

CPU	AMD Ryzen 5 2600 @ 3.4 GHz, 6 cores
RAM	16GB
GPU	NVIDIA GeForce RTX 2060
Operating System	Arch Linux
Kernel Version	6.3.1
Python	3.11.3

Table 3.2: AMD Ryzen desktop specification used in benchmarking Green Lantern.

CPU	Intel Haswell i5-4300U @ 2.9 GHz, 2 cores
RAM	8GB
GPU	Intel HD Graphics 4400
Operating System	Fedora Linux
Kernel Version	6.0.7
Python	3.11.0

Table 3.3: Intel Haswell laptop specification used in benchmarking Green Lantern.

The specific metrics used to evaluate the efficacy of Green Lantern are relative performance-loss and energy-savings, as done by Spiliopoulos, Kaxiras, and Keramidas (2011), Ge et al. (2007) and Kim et al. (2012). To obtain these a baseline variable is set by first running the benchmarks with no DVFS technique other than the default Linux *ondemand* governor, testing for specific raw metrics: time taken to complete the benchmark and electricity consumed throughout the benchmark’s execution. Then, the benchmarks are run again alongside Green Lantern. By measuring the difference in time taken to complete the benchmark the relative performance-loss can be calculated. Likewise, energy-savings are calculated by comparing the difference in energy consumed.

3.4 Objective 4

To assess that the developed technique is accessible to SMEs and consumers relative to other techniques.

In order to reduce carbon footprint as much as possible, a technique should aim to be used as widely as possible. In order to achieve this, barriers to its adoption should be minimised. Based on research by Ahmed (2018) and Ahmad et al. (2021) regarding barriers to adoption for green computing a framework for quantifying the accessibility of a green computing technique was designed, which can be found in table 3.4. Each DVFS technique discussed in chapter 2 is assessed for the presence of various barriers to adoption and given a numeric score for each barrier lacking in the technique, meaning the technique possesses a higher degree of accessibility. These scores are then compared with each other and with Green Lantern.

ID	Description
AC1	Concerns about cost.
AC2	Concerns about security.
AC3	Concerns about implementation difficulty.
AC4	Concerns about CPU compatibility.
AC5	Concerns about configurability.

Table 3.4: Framework for quantifying accessibility of green computing techniques.

Chapter 4

Implementation

This chapter details the implementation of Green Lantern. Firstly, a concrete set of requirements was produced from the systematic literature review found in chapter 2. Secondly, the details of the Green Lantern algorithm and the rationale behind decisions made in its development is presented. To aid in understanding the technique presented and to aid in its adoption as mentioned in chapter 2, both pseudocode and a flowchart of the algorithm are presented, as well as the source code supplied in the appendices. Thirdly, details of the configurability of Green Lantern are presented. Finally, the details behind the Linux governor-based DVFS strategy are presented and discussed.

4.1 Requirements

In order to fill the gap in the literature by implementing a novel carbon-aware DVFS technique, specific requirements in order to achieve this were extracted from chapter 2. These can be found in table 4.3.

Firstly, in order to be carbon-aware, as defined by Anderson et al. (2022), it needs to possess “clearly visible and clearly defined providence for the electricity that it consumes during runtime”. In order to achieve this the UK National Grid’s carbon intensity API is utilised. The API provides a variety of endpoints for various purposes such as statistics of a specific time-frame but only the Regional and National endpoints are required for the implementation of Green Lantern.

Ideally, carbon-aware software should also possess fine-grained control over the type of electricity that it consumes, for example the ability for the end-user to have fine-grained control over whether the program consumes solar energy, biofuel, or coal. While the carbon intensity API does have this capability this functionality is not implemented in Green Lantern because it is infeasible to implement correctly. This is due to the fact that all regions in the UK use a blend of electricity types. For an illustrative example, see tables 4.1 and 4.2. Despite being regions

that use some of the highest amounts of renewable energy in the UK (Rogers and Parson, 2019), varying degrees of non-renewable fuels are still used in these areas making such a fine degree of control impossible at the time of writing and is thus outside of Green Lantern’s scope.

Fuel Type	Percentage
Biomass	2.3%
Coal	0%
Imports	9.8%
Gas	17.3%
Nuclear	60.3%
Other	0%
Hydro	0.1%
Solar	1.9%
Wind	8.3%

Table 4.1: Breakdown of fuel usage of North West England on the 18th May 2023. Source: NationalGrid (2023).

Fuel Type	Percentage
Biomass	0.5%
Coal	0%
Imports	3.9%
Gas	0%
Nuclear	43.9%
Other	0%
Hydro	2.1%
Solar	2.3%
Wind	47.1%

Table 4.2: Breakdown of fuel usage of South Scotland on the 18th May 2023. Source: NationalGrid (2023).

Secondly, Green Lantern needs to utilise DVFS in order to scale CPU performance based on the amount of carbon intensity both nationally and regionally. Various approaches were assessed, such as Ge et al. (2007) which achieves significant results but only works on a specific CPU architecture. Or Spiliopoulos, Kaxiras, and Keramidas (2011) that possesses greater accessibility but creates security and further accessibility concerns due to requiring manual patching of the Linux kernel. Green Lantern addresses these concerns by using a DVFS strategy already built-in, though underutilised, to every Linux distribution: the Linux CPU governors. The code behind these governors is well-documented, audited for security and has been tested in a wide variety of production environments. Green Lantern makes use of these governors as DVFS policies to be used based on the carbon intensity forecast. Further details of this technique are presented in 4.4.

Thirdly, it is required to avoid adoption barriers and have compatibility with as many CPUs

as possible. Techniques such as Ge et al. (2007), though possessing great promise in terms of energy-savings, have not achieved a widespread decrease in carbon footprint because they are difficult to implement and need to be implemented on a per-CPU basis. As stated above, a strategy based around the Linux CPU governors solves this concern.

Fourthly, Green Lantern needs to be free and open-source. One of the highest barriers to adoption of green computing techniques found by Ahmad et al. (2021) to be concerns with cost, a barrier that can be partially abolished by making the technique free and providing the source code freely licensed.

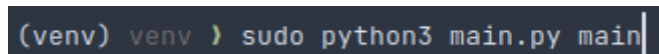
Finally, Green Lantern must be user-configurable. Another barrier to adoption identified by Ahmad et al. (2021) is the lack of services to address the specific needs of clients. In order to minimise the effect of this barrier Green Lantern provides options for the user to configure to their needs. As discussed in chapter 3, TOML is used for this to make this as accessible as possible.

ID	Description
R1	Continuously monitor the carbon intensity of the UK locality of a given server.
R2	Utilise DVFS to scale performance according to the local carbon intensity.
R3	Compatibility with both Intel and AMD CPUs.
R4	Free and open-source.
R5	User-configurable to allow users to cater GL to their specific needs.

Table 4.3: Green Lantern implementation requirements.

4.2 User Interface

Green Lantern uses a command-line interface. Considering that a barrier to adoption identified by both Ahmed (2018) *and* Ahmad et al. (2021) is *complexity* it could be argued that a GUI should be used for Green Lantern instead. However, while this would reduce the complexity in using Green Lantern would mean that it cannot be used on servers and other environments that do not possess a GUI. Rather, Green Lantern’s interface is explicitly designed to be as simple. A screenshot of the command used to start Green Lantern can be observed in figure 4.1. A screenshot of the program in operation can be observed in figure 4.2.



```
(venv) venv > sudo python3 main.py main
```

Figure 4.1: Screenshot of command used to start Green Lantern.

```

CI forecast within 25% or equal to specified limit...
Carbon intensity forecast: 268
CPU governor: powersave
=====
CI forecast less than or equal to 75% of specified limit...
Carbon intensity forecast: 240
CPU governor: conservative
=====

```

Figure 4.2: Screenshot of Green Lantern in operation.

4.3 Algorithm

Green Lantern runs continuously in the background, polling the Carbon Intensity API at a user-defined interval and then adjusts the CPU governor based on the results. The algorithm is expressed as the pseudocode found in 4.3 and as a flowchart in figure 4.4.

```

BEGIN
INPUT configuration
PARSE configuration
WHILE TRUE
    IF location IS SET
        GET carbon_intensity
    ELSE
        INPUT public_ip
        GET location
        GET carbon_intensity

    INPUT carbon_intensity_limit
    IF carbon_intensity <= (0.25 * carbon_intensity_limit)
        SET GOVERNOR = performance
    ELSE IF carbon_intensity <= (0.5 * carbon_intensity_limit)
        SET GOVERNOR = ondemand
    ELSE IF carbon_intensity <= (0.75 * carbon_intensity_limit)
        SET GOVERNOR = conservative
    ELSE carbon_intensity >= carbon_intensity
        SET governor = powersave

    INPUT poll_interval
    SLEEP(poll_interval)
END WHILE
END

```

Figure 4.3: Pseudocode representation of the Green Lantern algorithm.

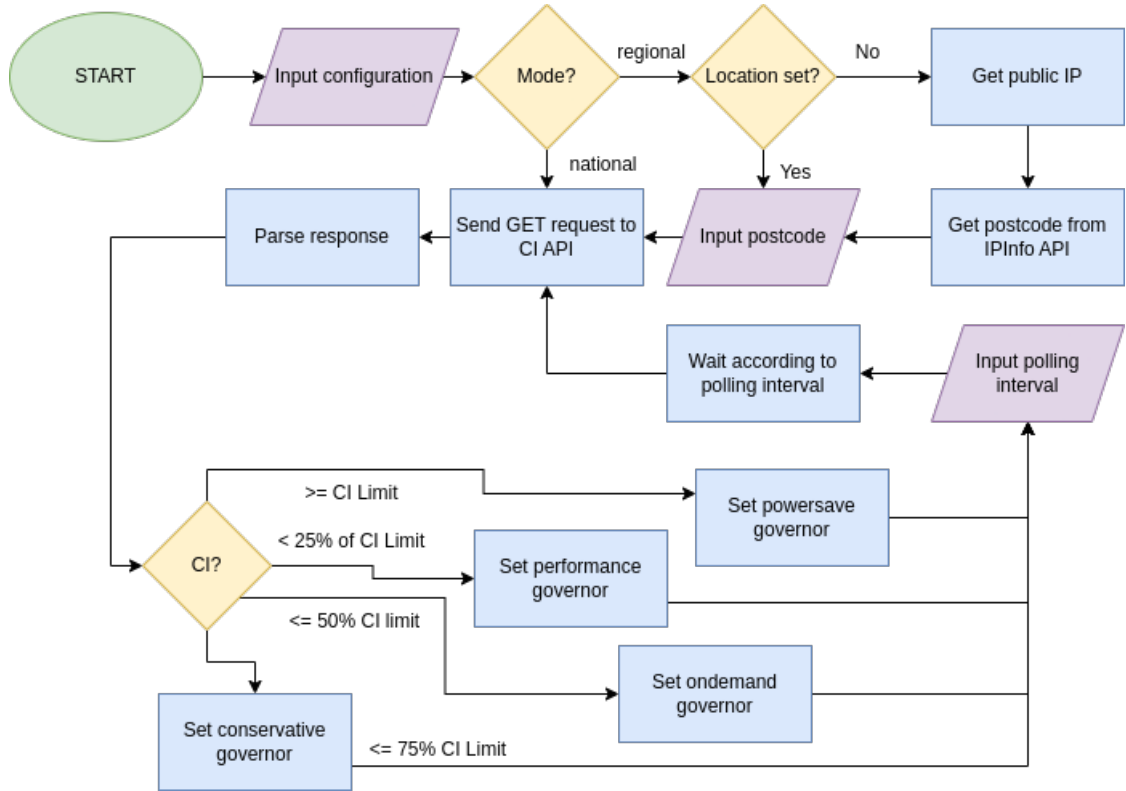


Figure 4.4: Flowchart representation of the Green Lantern algorithm.

4.4 DVFS

As mentioned in chapter 3, the strategy for DVFS in Green Lantern prioritises accessibility. Therefore, the in-built Linux CPU governors are used: *performance*, *ondemand*, *conservative* and *powersave*. A deeper explanation of these governors can be found in table 4.4. Using kernel governors minimises barriers to adoption, ensures a wide range of CPU compatibility and is more secure than experimental but unaudited DVFS methods, ensuring that Green Lantern is more suited to production environments compared to other techniques.

Governor Name	Description
performance	Statically sets CPU to highest frequency.
ondemand	Scales CPU frequency according to approximate system load via the Linux kernel scheduler periodically sampling CPU-usage statistics.
conservative	Similar to <i>ondemand</i> in that it scales CPU performance alongside system load but scales gradually rather than suddenly spiking performance, using less energy.
powersave	Statically sets CPU statically to the lowest frequency.

Table 4.4: Table containing a description of the Linux CPU governors used for Green Lantern.

4.5 Carbon-Awareness

In order to make Green Lantern carbon-aware it needs to monitor the carbon intensity both at a national and regional level. For this, the National Grid carbon intensity API is used. The API is sent a postcode using a GET request and returns a JSON response containing the information found in figures 4.5 and 4.6, examples of responses for the regional and national endpoints respectively. The code found in figure 4.7 parses this JSON response and extracts the required information from it to then be utilised by Green Lantern.

```
{
  "data": [
    {
      "regionid": 8,
      "dnoregion": "WPD West Midlands",
      "shortname": "West Midlands",
      "postcode": "GL52",
      "data": [
        {
          "from": "2023-05-17T16:00Z",
          "to": "2023-05-17T16:30Z",
          "intensity": {
            "forecast": 260,
            "index": "high"
          },
          "generationmix": [
            {
              "fuel": "biomass",
              "perc": 0.6
            },
            {
              "fuel": "coal",
              "perc": 0
            },
            {
              "fuel": "imports",
              "perc": 5.5
            },
            {
              "fuel": "gas",
              "perc": 65.7
            },
            {
              "fuel": "nuclear",
              "perc": 9
            },
            {
              "fuel": "other",
              "perc": 0
            },
            {
              "fuel": "hydro",
              "perc": 0
            },
            {
              "fuel": "solar",
              "perc": 13.9
            },
            {
              "fuel": "wind",
              "perc": 5.3
            }
          ]
        }
      ]
    }
  ]
}
```

Figure 4.5: Example API response for the regional carbon intensity API. Endpoint: `api.carbonintensity.org.uk/intensity`

```
{
  "data": [
    {
      "from": "2023-05-17T16:00Z",
      "to": "2023-05-17T16:30Z",
      "intensity": {
        "forecast": 178,
        "actual": 174,
        "index": "moderate"
      }
    }
  ]
}
```

Figure 4.6: Example API response for the national carbon intensity API. Endpoint: `api.carbonintensity.org.uk/regional/postcode/GL52`


```

def start(config):
    location = get_location(config)

    while True:
        ci_request = get_CI(location, config)
        ci_forecast = ci_request['data'][0]['data'][0]['intensity']['forecast']
        gov = determine_governor(config, ci_forecast)

        print("Carbon intensity forecast: " + str(ci_forecast))

        try:
            set_governor(gov)
            print("CPU governor: " + get_governor_name(gov))
            print("=====")

        except:
            print("Unable to adjust CPU governor. Please ensure you are using a compatible CPU.")

        time.sleep(config['poll_interval'])

```

Figure 4.7: Code used to parse the JSON response from carbon intensity API and adjust governor.

4.6 Configuration

In order to improve accessibility and reduce barriers to adoption, Green Lantern is configurable by the end-user according to their needs. There are several configuration options available to the user, which can be found in table 4.5.

Option	Values	Description
mode	'regional', 'national'	Determines whether to use the national carbon intensity or regional.
location	valid UK postcode	Allows user to explicitly set location rather than have it obtained from public IP address.
carbon_intensity_limit	positive in- teger	carbon intensity value in which to switch to the 'powersave' governor, saving the most energy.
poll_interval	positive in- teger	Determines the number of seconds to wait in-between rechecking the carbon intensity of the local area and adjusting the DVFS strategy accordingly.

Table 4.5: Configuration options available to the end-user.

Once implemented the configuration file looks as shown in figure 4.8.

```
[config]
mode = "regional"
location = "GL52"
carbon_intensity_limit = "350"
poll_interval = 10
```

Figure 4.8: Illustrative example of Green Lantern’s configuration file.

Chapter 5

Results & Analysis

This section details exactly how Green Lantern was benchmarked and evaluated against other techniques. It includes the rationale behind how it was tested and the design of the simulation of real-world workloads. Null and alternative hypotheses are then established and tested against the results of the benchmarks. Alongside testing for statistical significance, the results of Green Lantern in CPU-bound workloads, memory-bound workloads and accessibility are compared to the other techniques.

5.1 Experiment Design

As discussed in chapter 3, various experiments were designed to empirically benchmark Green Lantern and evaluate the results against a baseline and against other solutions. For these benchmarks, the carbon-intensity is set to fluctuate every 10 seconds randomly between 50 and 350 to simulate common fluctuations in carbon-intensity such as the passing of day and night. Benchmarking functionality was developed into Green Lantern, letting it utilise mock data for the purposes of testing. An *ondemand* baseline result is firstly established for each benchmark on both Intel and AMD and then tested against Green Lantern under the exact same conditions. The benchmarks were run with the specific configuration found in table 5.1. This configuration represents a realistic example of Green Lantern in practice.

Option	Value
mode	regional
location	GL52
carbon_intensity_limit	350
poll_interval	10

Table 5.1: Green Lantern configuration options used in benchmarking.

5.1.1 Hypotheses

Hypothesis H_0 : Green Lantern does not achieve energy-savings over the baseline. ($p \geq 0.05$).

Hypothesis H_1 : Green Lantern achieves statistically significant energy-savings over the baseline. ($p < 0.05$).

5.2 Baseline

This section provides the results of running the memory-bound `libquantum` and CPU-bound `calculix` benchmarks tested against the Linux *ondemand* governor to set a baseline to test Green Lantern against.

5.2.1 Intel

Memory-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	49778.142	3954.861
2	50301.966	3958.939
3	49524.480	3966.565
Mean	49868.196	3960.122

Table 5.2: Baseline `libquantum` benchmark results on Intel.

CPU-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	47804.276	4288.519
2	47817.382	4315.033
3	47880.701	4273.315
Mean	47834.120	4292.289

Table 5.3: Baseline `calculix` benchmark results on Intel.

5.2.2 AMD

Memory-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	12821.195	2158.495
2	13629.576	2148.038
3	13336.108	2173.905
Mean	13262.293	2160.146

Table 5.4: Baseline `libquantum` benchmark results on AMD.

CPU-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	41130.103	2849.812
2	40648.200	2845.042
3	41120.914	2844.678
Mean	40966.406	2846.511

Table 5.5: Baseline calculix benchmark results on AMD.

5.3 Green Lantern

5.3.1 Intel

Memory-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	43632.157	4619.437
2	44666.428	4612.475
3	44878.653	4611.392
Mean	44392.413	4614.435

Table 5.6: Green Lantern libquantum benchmark results on Intel.

CPU-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	47939.961	5276.755
2	47917.565	5054.180
3	47615.015	5052.760
Mean	47824.180	5127.898

Table 5.7: Green Lantern calculix benchmark results on Intel.

5.3.2 AMD

Memory-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	10471.266	2112.102
2	8610.922	2053.904
3	11667.270	2109.856
Mean	10249.819	2091.954

Table 5.8: Green Lantern libquantum benchmark results on AMD.

CPU-Bound Workload

Iteration	Energy (J)	Execution Time (s)
1	41368.114	2843.509
2	41479.461	2843.841
3	42385.365	2848.798
Mean	41744.313	2845.383

Table 5.9: Green Lantern calculix benchmark results on AMD.

5.4 Accessibility

In the interest of maximising adoption of a green computing approach, accessibility has been prioritised in the development of Green Lantern. Based on a review of the literature, a list of barriers to adoption of green computing techniques was extracted in chapter 2. The barriers driven by accessibility issues were then compiled to devise a framework to empirically assess the accessibility of a green computing technique. The accessibility of each technique reviewed in section 2 was then quantified using this framework and compared to Green Lantern. A No (N) grants a score of 1, indicating no concerns about the barrier to adoption in question.

5.4.1 CPU-MISER

Barrier to Adoption	Present (Y/N)
AC1	Y
AC2	N
AC3	Y
AC4	Y
AC5	N
Total	2

Table 5.10: Table summarising the accessibility of CPU-MISER.

5.4.2 eDVFS

Barrier to Adoption	Present (Y/N)
AC1	Y
AC2	N
AC3	Y
AC4	Y
AC5	N
Total	2

Table 5.11: Table summarising the accessibility of eDVFS.

5.4.3 GG

Barrier to Adoption	Present (Y/N)
AC1	N
AC2	Y
AC3	Y
AC4	N
AC5	N
Total	3

Table 5.12: Table summarising the accessibility of GG.

5.4.4 FoREST

Barrier to Adoption	Present (Y/N)
AC1	Y
AC2	N
AC3	Y
AC4	N
AC5	N
Total	3

Table 5.13: Table summarising the accessibility of FoREST.

5.4.5 Green Lantern

Barrier to Adoption	Present (Y/N)
AC1	N
AC2	N
AC3	N
AC4	N
AC5	N
Total	5

Table 5.14: Table summarising the accessibility of Green Lantern.

5.5 Analysis

This section analyses the results of the benchmarking, performs statistical analysis in order to determine the significance of the results, compares results with the solutions discussed in chapter 2, discusses accessibility and finally finishes with a discussion of the implications of the results.

5.5.1 Memory-Bound Workloads

Upon completion of the benchmarks, an independent sample T-test needs to be performed in order to assess the significance of the results. In the memory-bound workload benchmark,

Green Lantern achieved a statistically significant 23% decrease in energy-usage on AMD and a statistically significant 11% decrease on Intel, meaning the null hypothesis can be rejected. A table showing the t-statistic and p-values of the memory-bound workload for both Intel and AMD can be found in table 5.15. These results are expected given that all other DVFS techniques discussed also achieved significant energy-savings on memory-bound workloads due to the fact that the CPU has significantly more time waiting for the memory as the main bottleneck and it is this principle that underpins most DVFS techniques.

Regarding performance-loss, the benchmarks were completed 3% faster on AMD with Green Lantern compared to baseline. Due to the inherent variability of carbon intensity, alongside the randomness introduced into the benchmarking process in order to simulate this variability, it is likely that this decrease occurred due to the performance governor being utilised often in the benchmarks, resulting in a faster completion. Despite this, significant energy-savings were made. The older Haswell Intel CPU, on the other hand, experienced a significant 16% drop in performance. The reason for this is almost certainly due to the fact that the AMD CPU has six-cores compared to the Intel's two cores. Downscaling CPU frequency affects single-core performance more than multi-core performance, making CPUs with lower cores more vulnerable to performance-loss when using DVFS techniques.

On AMD, Green Lantern achieved comparable results to the techniques of Ge et al. (2007), Kim et al. (2012) and Halimi et al. (2013) in terms of energy-savings and outperformed these techniques in terms of performance-loss. However, on Intel Green Lantern achieved inferior results with only an 11% decrease compared to the 15% decrease reported by Halimi et al. (2013), the lowest among them. Likewise, Green Lantern has inferior performance to these techniques on Intel with a 16% drop in performance on average compared to even the worst of the techniques in this regard, Spiliopoulos, Kaxiras, and Keramidas (2011) with a 10% performance-loss.

CPU	T-Statistic	P-Value
AMD	3.2742014801904866	0.030667594793414656
Intel	12.224326691909097	0.00025711098725052114

Table 5.15: T-statistics and P-values for the memory-bound workload benchmark.

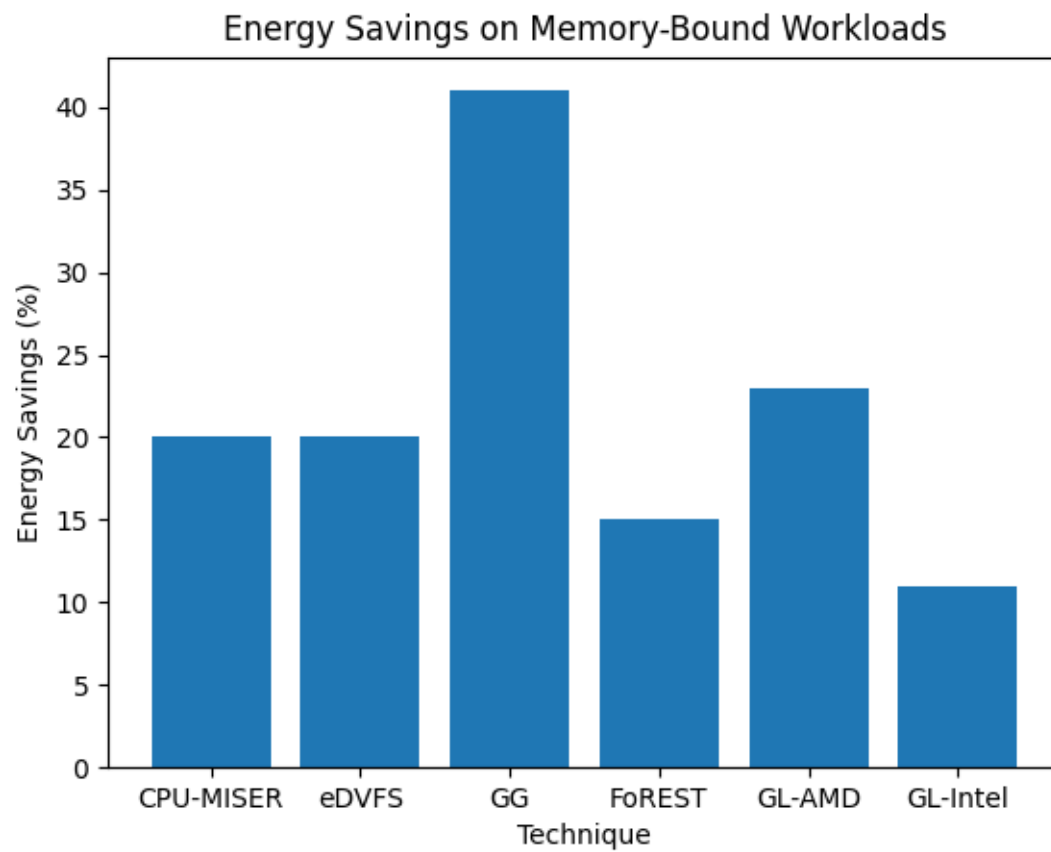


Figure 5.1: Comparison of energy-savings on memory-bound workloads.

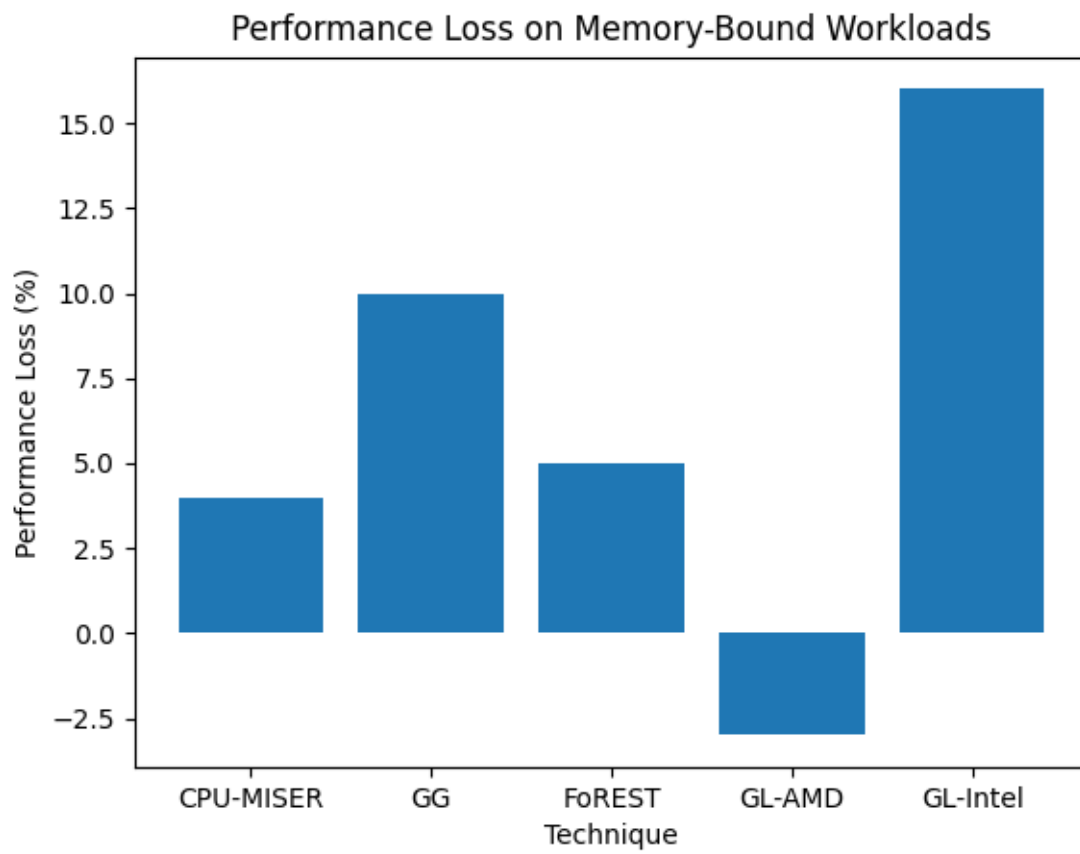


Figure 5.2: Comparison of performance-loss on memory-bound workloads.

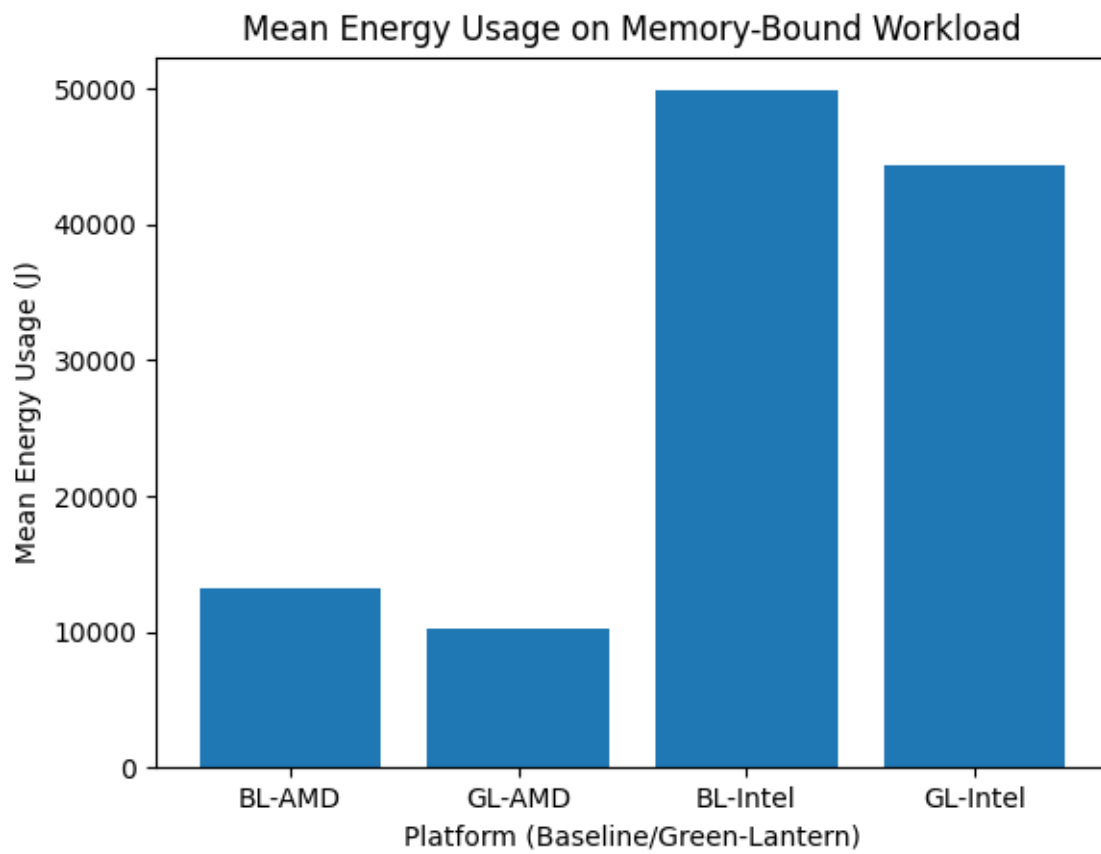


Figure 5.3: Comparison of mean energy-usage on memory-bound workloads.

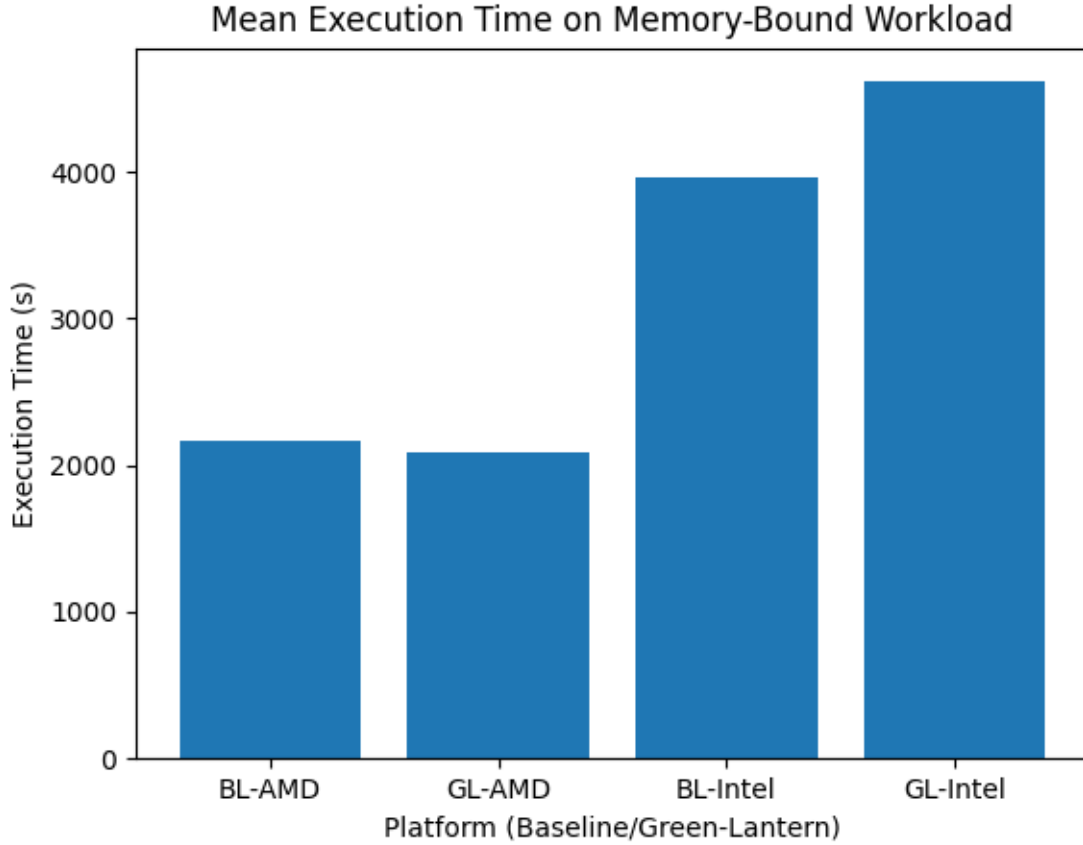


Figure 5.4: Comparison of mean execution time on memory-bound workloads.

5.5.2 CPU-Bound Workloads

In CPU-bound workloads Green Lantern does not achieve statistically significant energy-savings. On AMD it achieved a 1.89% *increase* in energy consumption and on Intel a mere 0.02% decrease. T-statistics and P-values for these results can be found in table 5.16, resulting in the null hypothesis being accepted for CPU-bound workloads due to neither P-value being beneath 0.05. An increase in energy consumption when using DVFS in CPU-bound workloads was also found to be the case when benchmarking Halimi et al. (2013), meaning that Green Lantern remains comparable to other solutions despite lackluster results on the CPU-bound benchmark.

Due to DVFS techniques primarily functioning by taking advantage of ‘slack’ in the CPU, for instance when accessing memory, it is to be expected that energy-savings are therefore minimal at best and non-existent at worst in CPU-bound workloads where the CPU is the main bottleneck. Furthermore, performance is impacted much more significantly than with memory-bound workloads. This is due to the fact that reducing CPU frequency while waiting for CPU to finish accessing the RAM hinders performance less than reducing CPU frequency while the CPU is the primary determinant of execution speed. Due to none of the discussed techniques providing specific benchmark results to be empirically tested against, Green Lantern cannot be

compared to these techniques.

CPU	T-Statistic	P-Value
AMD	-2.1651127168155186	0.09633576743024484
Intel	0.09253962141529644	0.9307188297578037

Table 5.16: T-statistics and P-values for the CPU-bound workload benchmark.

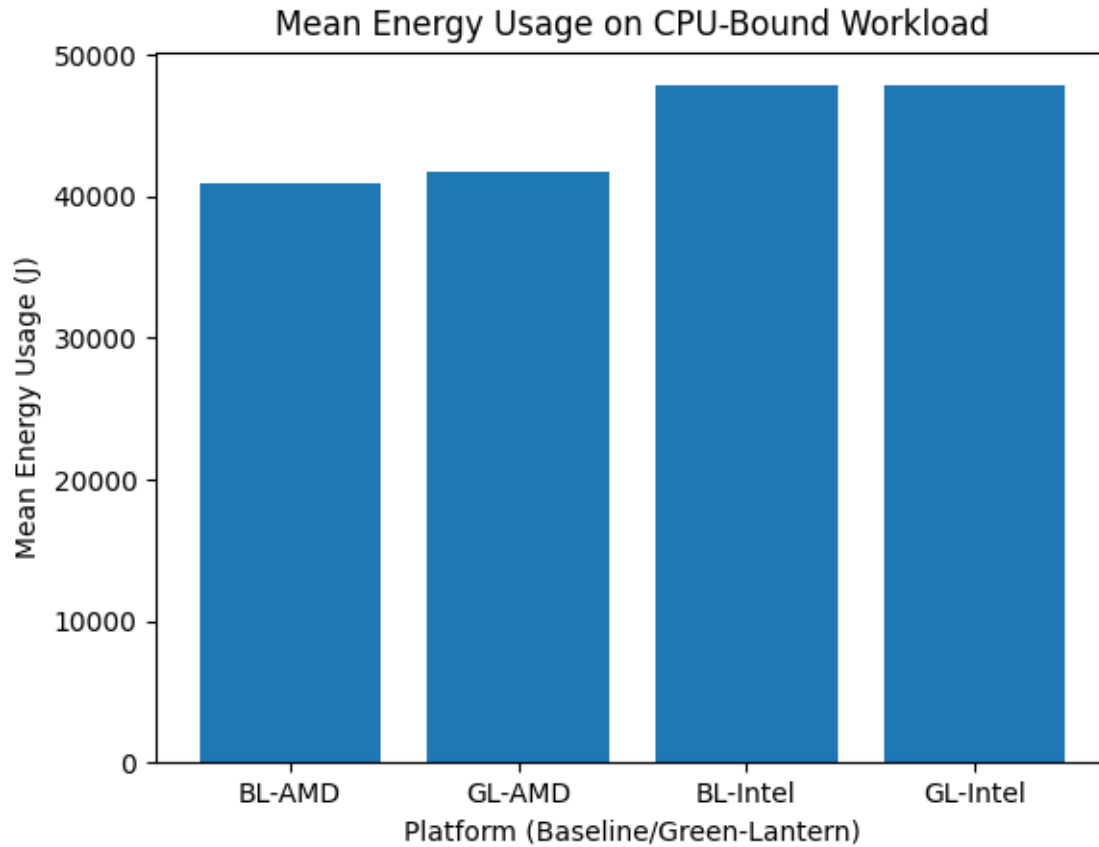


Figure 5.5: Comparison of mean energy-usage on CPU-bound workloads.

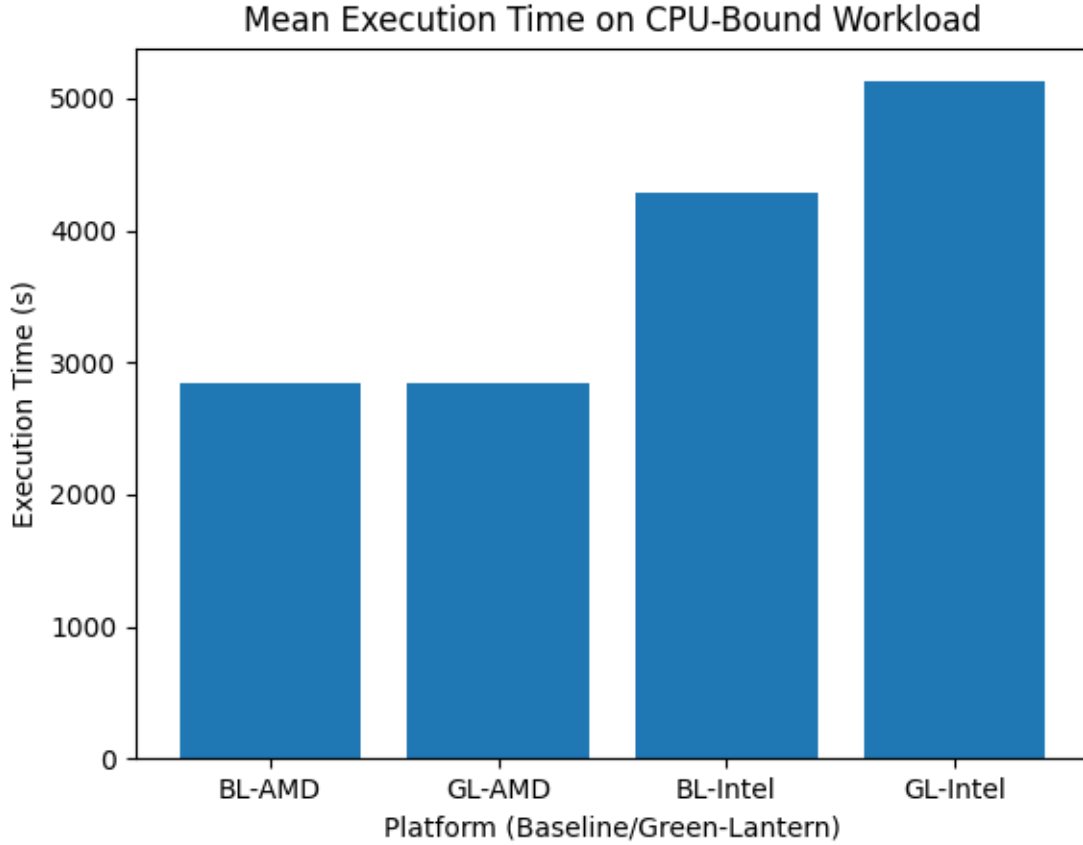


Figure 5.6: Comparison of mean execution time on CPU-bound workloads.

5.5.3 Accessibility

Using the framework for quantifying the accessibility of green computing techniques outlined in chapter 3, it was found that Green Lantern is the most accessible among them when keeping in mind the barriers to adoption identified by authors such as Ahmed (2018) and Ahmad et al. (2021). This comes as no surprise given that accessibility was kept as a priority, even at the expense of greater energy-savings, in the design of Green Lantern. Comparatively, the other DVFS techniques evaluated in this dissertation did not consider accessibility in their design other than allowing for a user-defined performance-loss constraint in terms of configurability. Indeed, despite offering its own configuration options Green Lantern does not offer a user-defined performance-loss constraint. The reason for this is that it uses a less sophisticated DVFS strategy in order to maximise compatibility across various CPUs and systems; developing a model specialised enough to accurately predict and constrain performance-loss across CPU brands and architectures is outside of the scope of Green Lantern and is a very large undertaking. An overview of differences in accessibility between techniques can be seen in figure 5.7.

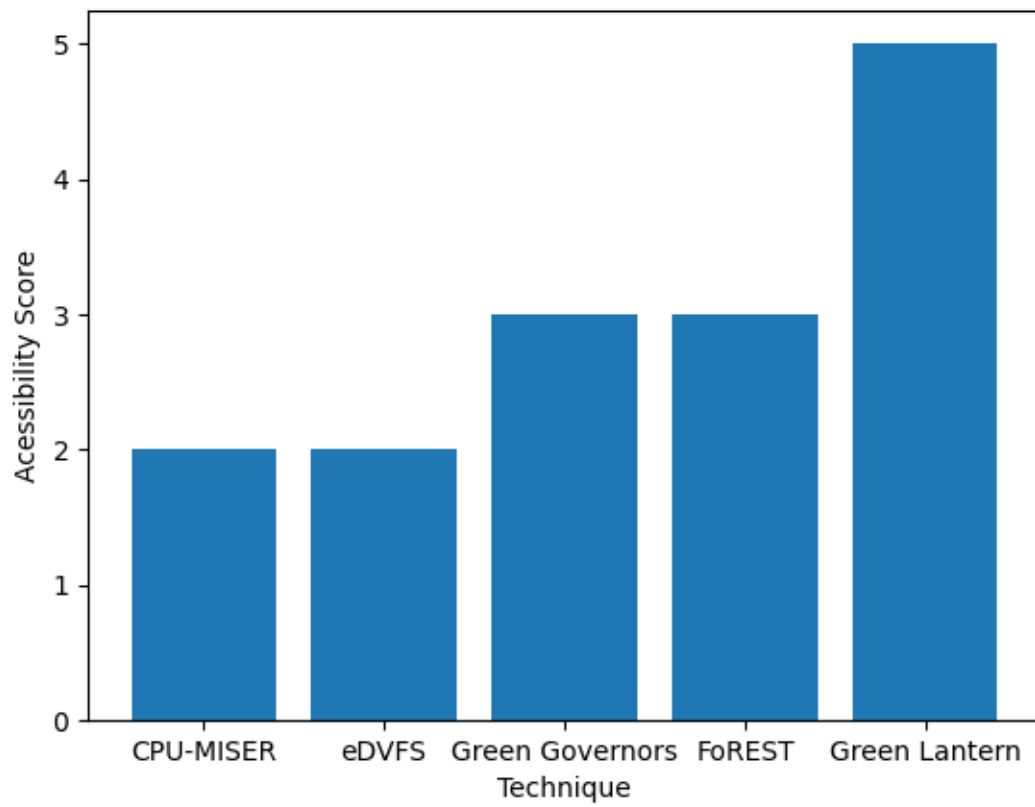


Figure 5.7: Comparison of accessibility between techniques.

Chapter 6

Conclusion

This section wraps up the dissertation with a discussion of the relative strengths and limitations of Green Lantern and provides recommendations for future research.

6.1 Strengths

Green Lantern provides a novel, carbon-aware DVFS solution that is compatible with both Intel and AMD CPUs, and theoretically, all CPUs which support the Linux kernel governors. Alongside this, it allows end-users to configure their location, whether to use national or regional carbon intensity forecasts and to set a specific carbon limit with which to scale CPU frequency alongside. Due to it being designed specifically to combat barriers to adoption of green computing technologies, it comes without security concerns and is purposely built to be low-cost to implement. This is due to the fact that it uses audited and hardened code already present in the Linux kernel for DVFS rather than relying on patching the Linux kernel, as done by Halimi et al. (2013) and doesn't rely on CPU-specific implementations of DVFS strategies such as Ge et al. (2007).

Furthermore, Green Lantern is free and open-source allowing its code and technique to be utilised by any interested party, significantly reducing barriers to adoption relative to other techniques. Due to its reliance on functionality integral to the Linux kernel, the maintenance of Green Lantern's DVFS strategy is performed by the thousands of developers working on the Linux kernel. Any further improvements made in the kernel's various CPU governors will also yield further energy-savings or further performance boosts for Green Lantern, only requiring a kernel upgrade.

Overall, Green Lantern has successfully met the objectives outlined in chapter 1, possesses all of the requirements outlined in chapter 4 and represents a successful proof-of-concept for a novel combination of carbon-aware software techniques with DVFS in order to reduce the carbon footprint of computer, thus filling in the gap in the literature identified in chapter 2 and

opening new avenues of research.

6.2 Limitations

Despite offering a robust and accessible carbon-aware DVFS solution, Green Lantern as an initial proof-of-concept does have a few limitations. Firstly, its DVFS strategy is less sophisticated than the approaches taken by other researchers. For instance, it does not perform real-time modelling of the CPU in order to more precisely scale frequency and voltage on LLC misses. Though this decision was made to improve CPU-compatibility it does decrease the potential energy-saving capabilities of Green Lantern. Green Lantern demonstrates the potential for carbon-aware software to revolutionise the computing industry as a proof-of-concept but it does not provide the most specialised or sophisticated DVFS strategy and should not be used for this purpose.

Secondly, though a deliberate limitation it is a limitation nonetheless, Green Lantern is only compatible with Linux as it is the operating-system used by most servers. Thirdly and finally, though the efficacy of the Green Lantern technique was found to be statistically significant across two devices it should be subject to more rigorous testing in production environments with a greater diversity of CPUs and other hardware configurations. Ideally, it should be evaluated in a real data-centre. However, at the time of writing, this was not possible to achieve.

6.3 Future Research

A curious finding obtained from the evaluation of Green Lantern was that in certain contexts, such as having more cores, greater energy-savings can actually be achieved not by downscaling CPU performance but by upscaling it, a finding also backed up by Kim et al. (2012). Future research should improve upon Green Lantern by using its carbon-aware capabilities in combination with a more sophisticated DVFS technique. Ideally, a technique that can intelligently model when it is more beneficial in terms of energy-savings and performance to boost or decrease performance and adjust performance accordingly, something lacking in current DVFS research. Though in doing this it remains important to ensure that the technique is designed around accessibility and compatible with as many CPUs as possible. In working with a DVFS technique that doesn't rely on the Linux kernel it may be possible to expand compatibility to other operating systems such as Windows and OSX too, Green Lantern on the other hand targets specifically Linux as it makes up the vast majority of servers in use today, which in turn use the majority of electricity and subsequent carbon footprint. To expand on this even further, research should not only focus on monitoring CPU energy-usage to improve DVFS techniques but a technique to monitor the real-time carbon intensity of the computer itself would allow techniques to be developed that bring even further reductions in carbon footprint with less compromises on performance.

Bibliography

- Ahmad, A., Khan, S. U., Khan, H. U., Khan, G. M., and Ilyas, M. (2021). ‘Challenges and Practices Identification via a Systematic Literature Review in the Adoption of Green Cloud Computing: Client’s Side Approach’. *IEEE Access*, 9, pp. 81828–81840.
- Ahmed, A. I. (2018). ‘Understanding the factors affecting the adoption of green computing in the Gulf Universities’. *International Journal of Advanced Computer Science and Applications*, 9.(3), pp. 304–3011.
- Alasdair, B., Ruff, L., Kelloway, J., MacMillan, F., and Rogers, A. (2020). ‘Carbon Intensity Methodology: Regional Carbon Intensity’. *NationalGridESO*, Available at: <https://github.com/carbon-intensity/methodology/raw/master/Regional%20Carbon%20Intensity%20Forecast%20Methodology.pdf> (Accessed: May 4, 2023).
- Alvi, H. M., Sahar, H., Bangash, A. A., and Beg, M. O. (2017). ‘Ensights: A tool for energy aware software development’. *2017 13th International Conference on Emerging Technologies (ICET)*. IEEE, pp. 1–6.
- Anderson, T., Belay, A., Chowdhury, M., Cidon, A., and Zhang, I. (2022). ‘Treehouse: A case for carbon-aware datacenter software’. *arXiv preprint arXiv:2201.02120*,
- Andrae, A. S. (2020). ‘New perspectives on internet electricity use in 2030’. *Engineering and Applied Science Letters*, 3.(2), pp. 19–31.
- Barnes, P. W. et al. (2019). ‘Ozone depletion, ultraviolet radiation, climate change and prospects for a sustainable future’. *Nature Sustainability*, 2.(7), pp. 569–579.
- Belkhir, L. and Elmeligi, A. (2018). ‘Assessing ICT global emissions footprint: Trends to 2040 & recommendations’. *Journal of cleaner production*, 177, pp. 448–463.
- Bharany, S. et al. (2022). ‘A systematic survey on energy-efficient techniques in sustainable cloud computing’. *Sustainability*, 14.(10), p. 6256.
- Brown, R. E. et al. (2007). *Report to congress on server and data center energy efficiency: Public law 109-431*. Tech. rep. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).
- Dalvi-Esfahani, M. and Nilashi, M. (2022). ‘Analyzing the Barriers Affecting the Adoption of Green Computing: A Hybrid Method Approach’.
- EIA, U. (2021a). *How much carbon dioxide is produced per kilowatthour of US electricity generation*. Available at: <https://www.eia.gov/tools/faqs/faq.php?id=74&t=11> (Accessed: Mar. 21, 2023).

-
- EIA, U. (2021b). *Total Energy: Primary Energy Consumption by Source*. Available at: <https://www.eia.gov/totalenergy/data/browser/index.php?tbl=T01.03#/f=A&start=2007&end=2021&charted=12-13> (Accessed: Mar. 21, 2023).
- Ge, R., Feng, X., Feng, W.-c., and Cameron, K. W. (2007). ‘Cpu miser: A performance-directed, run-time system for power-aware clusters’. *2007 International Conference on Parallel Processing (ICPP 2007)*. IEEE, pp. 18–18.
- Group, S. R. (2021). *Amazon, Microsoft & Google Grab the Big Numbers - But the Rest of Cloud Market Still Grows by 27%*. Available at: <https://www.srgresearch.com/articles/amazon-microsoft-google-grab-the-big-numbers-but-rest-of-cloud-market-still-grows-by-27> (Accessed: Apr. 8, 2023).
- Gupta, U. et al. (2021). ‘Chasing carbon: The elusive environmental footprint of computing’. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, pp. 854–867.
- Hales, S., Kovats, S., Lloyd, S., Campbell-Lendrum, D., and Salud, O. M. de la (2014). *Quantitative risk assessment of the effects of climate change on selected causes of death, 2030s and 2050s*. World Health Organization. ISBN: 9789241507691.
- Halimi, J.-P. et al. (2013). ‘Reactive dvfs control for multicore processors’. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. IEEE, pp. 102–109.
- Hoffmann, R., Dimitrova, A., Muttarak, R., Crespo Cuaresma, J., and Peisker, J. (2020). ‘A meta-analysis of country-level studies on environmental change and migration’. *Nature Climate Change*, 10.(10), pp. 904–912.
- IPCC (2022). *IPCC: Climate Change 2022, Mitigation of Climate Change, Summary for Policymakers*, pp. 8–9. Available at: https://report.ipcc.ch/ar6wg3/pdf/IPCC_AR6_WGIII_SummaryForPolicymakers.pdf (Accessed: Feb. 15, 2022).
- Jayalath, J., Chathumali, E., Kothalawala, K., and Kuruwitaarachchi, N (2019). ‘Green cloud computing: A review on adoption of green-computing attributes and vendor specific implementations’. *2019 International Research Conference on Smart Computing and Systems Engineering (SCSE)*. IEEE, pp. 158–164.
- Jiang, Y., Adams, B., and German, D. M. (2013). ‘Will my patch make it? and how fast? case study on the linux kernel’. *2013 10th Working conference on mining software repositories (MSR)*. IEEE, pp. 101–110.
- Kazandjieva, M., Heller, B., Gnawali, O., Hofer, W., and Kozyrakis, P. (2011). ‘Software or hardware: The future of green enterprise computing’. *Computer Science Technical Report CSTR*, 2.
- Kim, S.-g., Choi, C., Eom, H., Yeom, H. Y., and Byun, H. (2012). ‘Energy-centric DVFS controlling method for multi-core platforms’. *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, pp. 685–690.
- 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), p. 161.
- Kurp, P. (2008). ‘Green computing’. *Communications of the ACM*, 51.(10), pp. 11–13.

-
- Limited, C. E. (2023). *Comtec Power: Datacentre & Server Room Solutions*. Available at: <https://www.comtecpower.com/> (Accessed: Mar. 21, 2023).
- Lynas, M., Houlton, B. Z., and Perry, S. (Nov. 2021). ‘Greater than 99% consensus on human caused climate change in the peer-reviewed scientific literature’. *Environmental Research Letters*, 16 (11). ISSN: 17489326. doi: 10.1088/1748-9326/ac2966.
- Mack, C. A. (2011). ‘Fifty years of Moore’s law’. *IEEE Transactions on semiconductor manufacturing*, 24.(2), pp. 202–207.
- Malhi, Y. et al. (2020). ‘Climate change and ecosystems: Threats, opportunities and solutions’. *Philosophical Transactions of the Royal Society B*, 375.(1794), p. 20190104.
- Masanet, E. R., Brown, R. E., Shehabi, A., Koomey, J. G., and Nordman, B. (2011). ‘Estimating the energy use and efficiency potential of US data centers’. *Proceedings of the IEEE*, 99.(8), pp. 1440–1453.
- Mayhew, P. J., Jenkins, G. B., and Benton, T. G. (2008). ‘A long-term association between global temperature and biodiversity, origination and extinction in the fossil record’. *Proceedings of the Royal Society B: Biological Sciences*, 275.(1630), pp. 47–53.
- NationalGrid (2023). *Carbon Intensity API*. Available at: <https://www.carbonintensity.org.uk/> (Accessed: May 18, 2023).
- Niles, S. and Donovan, P. (2008). ‘Virtualization and cloud computing: Optimized power, cooling, and management maximizes benefits’. *White paper*, 118.
- Pallipadi, V. and Starikovskiy, A. (2006). ‘The ondemand governor’. *Proceedings of the linux symposium*. Vol. 2. 00216, pp. 215–230.
- Powell, J. (Dec. 2017). ‘Scientists Reach 100% Consensus on Anthropogenic Global Warming’. *Bulletin of Science, Technology and Society*, 37 (4), pp. 183–184. ISSN: 15524183. doi: 10.1177/0270467619886266.
- Priestley, R. K., Heine, Z., and Milfont, T. L. (2021). ‘Public understanding of climate change-related sea-level rise’. *Plos one*, 16.(7), e0254348.
- Rocque, R. J. et al. (2021). ‘Health effects of climate change: an overview of systematic reviews’. *BMJ open*, 11.(6), e046333.
- Rogers, A. and Parson, O. (2019). *GridCarbon: A smartphone app to calculate the carbon intensity of the GB electricity grid*.
- Ruth, S. (2011). ‘Reducing ICT-related carbon emissions: an exemplar for global energy policy?’ *IETE technical review*, 28.(3), pp. 207–211.
- Satyanarayanan, M. (2017). ‘The emergence of edge computing’. *Computer*, 50.(1), pp. 30–39.
- Shehabi, A. et al. (2016). ‘United states data center energy usage report’.
- Spiliopoulos, V., Kaxiras, S., and Keramidas, G. (2011). ‘Green governors: A framework for continuously adaptive dvfs’. *2011 International Green Computing Conference and Workshops*. IEEE, pp. 1–8.
- Springmann, M. et al. (May 2016). ‘Global and regional health effects of future food production under climate change: A modelling study’. *The Lancet*, 387 (10031), pp. 1937–1946. ISSN: 1474547X. doi: 10.1016/s0140-6736(15)01156-3.

-
- Sriram, G. (2022). ‘Green cloud computing: an approach towards sustainability’. *International Research Journal of Modernization in Engineering Technology and Science*, 4.(1), pp. 1263–1268.
- Zhao, D. and Zhou, J. (2022). ‘An energy and carbon-aware algorithm for renewable energy usage maximization in distributed cloud data centers’. *Journal of Parallel and Distributed Computing*, 165, pp. 156–166.

Appendices

Appendix A

Source Code

A.1 main.py

```
import glutils
import argparse

"""
Main function to be used in real-world as opposed to
↪ benchmarking. Triggers initiation of Green Lantern.
"""

def main(config):
    if glutils.get_cpu_compatibility():
        glutils.start(config)
    else:
        print("Incompatible CPU. Please note that Green
        ↪ Lantern works only with AMD and Intel CPUs.")

"""
Initiates benchmarking, Green Lantern starts but uses
↪ randomised carbon-intensity data instead of real data.
"""

def benchmark(config):
    if glutils.get_cpu_compatibility():
        glutils.start_benchmark(config)
    else:
        print("Incompatible CPU. Please note that Green
        ↪ Lantern works only with AMD and Intel CPUs.")
```

```

"""
Parse command-line arguments. Main starts Green Lantern for
    ↪ real-world use, benchmark starts
Green Lantern with mock data.
"""
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('command', choices=['main', 'benchmark'
    ↪ ''])
    args = parser.parse_args()

    try:
        config = glutils.get_config("./config.toml")['config']

        if config is not None:
            if args.command == 'main':
                main(config)
            elif args.command == 'benchmark':
                benchmark(config)
    except KeyboardInterrupt:
        exit()
    except Exception as e:
        print("Please ensure that you have a valid
        ↪ configuration file in the local directory.")
        print(e)
        exit()

```

A.2 glutils.py

```

import toml
import cpuinfo
import requests
import ipinfo
import time
import random
from cpufreq import cpuFreq

supported_cpus = ["AMD", "INTEL"]

```

```

# To be initialised based on what's supported by system
cpu_governors = {}
current_governor = 0

# Keeps track of governor initiation
gl_init = False

"""
Get type of CPU of system.
"""
def get_cpu_brand():
    return cpuinfo.get_cpu_info()['brand_raw']

"""
Check if CPU is supported by Green Lantern, return true if so,
    ↪ false if not.
"""
def get_cpu_compatibility():
    brand_raw = get_cpu_brand()

    for cpu in supported_cpus:
        if cpu.upper() in brand_raw.upper():
            return True

    return False

"""
Open the configuration file stored in path.
"""
def get_config(path):
    with open(path, 'r') as f:
        return toml.load(f)

"""
Get the national or regional carbon-intensity based on which
    ↪ options set in the config.
"""

```

```

def get_CI(postcode, config):
    if postcode is None:
        return get_national_CI()
    elif len(config['location']) > 0:
        return get_regional_CI(postcode)

"""
Get random carbon-intensity forecast, used for benchmarking
→ purposes.
"""

def get_mock_CI():
    return random.randint(50, 350)

"""
Get regional carbon-intensity forecast for the postcode
→ location passed into the function.
Return JSON response.
"""

def get_regional_CI(postcode):
    headers = {
        'Accept': 'application/json'
    }

    try:
        return requests.get(f'https://api.carbonintensity.org.
            → uk/regional/postcode/{postcode}', params={},
                           headers=headers).json()

    except:
        print("Unable to connect to Carbon Intensity API.
            → Please check that you have a working internet
            → connection.")

"""
Get national carbon-intensity forecast for the UK.
"""

def get_national_CI():
    headers = {
        'Accept': 'application/json'
    }

```

```

    }

    try:
        return requests.get('https://api.carbonintensity.org.
            ↪ uk/intensity', params={}, headers=headers).json()
    except:
        print("Unable to connect to Carbon Intensity API.
            ↪ Please check that you have a working internet
            ↪ connection.")

"""
Get public IP address of server running Green Lantern in order
    ↪ to get a rough estimate of location.
"""
def get_public_ip():
    return requests.get('https://api.ipify.org').content.
        ↪ decode('utf8')

"""
Get postcode based on public IP address using the IPInfo API.
"""
def get_postcode(ip):
    access_token = 'be7b47304d7cee'
    handler = ipinfo.getHandler(access_token)
    details = handler.getDetails(ip)
    return details.postal

"""
Get location of computer/server, get from configuration if
    ↪ provided, get from public IP if not.
"""
def get_location(config):
    if len(config['location']) == 0 or config['location'] is
        ↪ None:
        print("Identifying location...")

        try:
            location = get_postcode(get_public_ip())

```

```

        if len(location) > 0 and location is not None:
            print("Identified location: " + location)
            return location
    except KeyboardInterrupt:
        exit()
    except:
        print(
            "Unable to determine location. Please ensure
            ↳ that you have a valid internet connection
            ↳ or input your "
            "postcode into the location item within the
            ↳ config.")
else:
    return config['location']

"""
Get governor's weight based on governor name.
"""
def get_governor_weight(governor):
    match governor:
        case "powersave":
            return 1
        case "conservative":
            return 2
        case "ondemand":
            return 3
        case "performance":
            return 4

"""
Get governor name based on governor weight.
"""
def get_governor_name(weight):
    match weight:
        case 1:
            return "powersave"
        case 2:
            return "conservative"

```

```

        case 3:
            return "ondemand"
        case 4:
            return "performance"

"""
Sort unsorted dictionary.
"""
def sort_dict(unsorted_dict):
    return dict(sorted(unsorted_dict.items()))

"""
Get governor currently in use by CPU.
"""
def get_current_governor(governors_dict):
    tmp_gov = next(iter(governors_dict.values()))

    for weight, gov in governors_dict.items():
        if gov != tmp_gov:
            return None

    return get_governor_weight(tmp_gov)

"""
Get governors supported by CPU and store in global dictionary.
"""
def init_governors():
    global cpu_governors
    global current_governor

    cpu = cpuFreq()
    tmp_cpu_governors = {}

    for gov in cpu.available_governors:
        if get_governor_weight(gov) is not None:
            tmp_cpu_governors[get_governor_weight(gov)] = gov

    cpu_governors = sort_dict(tmp_cpu_governors)

```

```

    current_governor = get_current_governor(cpu.get_governors
        ↪ ())

"""
Scale up to governor with +1 weight to boost CPU frequency.
"""
def upscale_governor():
    if gl_init is False or current_governor == 0:
        init_governors()

    cpu = cpuFreq()

    try:
        cpu.set_governors(cpu_governors[current_governor + 1])
        print("Carbon intensity within acceptable limit,
            ↪ increasing CPU performance...")
    except:
        print("Failed to increase CPU performance. Please
            ↪ check that you are running with sudo privileges."
            ↪ )

"""
Scale down to governor with -1 weight to decrease CPU
    ↪ frequency.
"""
def downscale_governor():
    if gl_init is False or current_governor == 0:
        init_governors()

    cpu = cpuFreq()

    try:
        cpu.set_governors(cpu_governors[current_governor - 1])
        print("Carbon intensity outside of acceptable limit,
            ↪ decreasing CPU performance...")
    except:
        print("Failed to decrease CPU performance. Please
            ↪ check that you are running with sudo privileges."
            ↪ )

```

```

"""
Set governor based on arbitrary weight.
"""
def set_governor(governor_weight):
    global current_governor

    if gl_init is False or current_governor == 0:
        init_governors()

    cpu = cpuFreq()

    try:
        # cpu.set_governors(cpu_governors[governor_weight])
        current_governor = governor_weight
    except:
        print("Failed to set specified governor. Please check
        ↳ that you are running with sudo privileges.")

"""
Determine which CPU governor to be used at various carbon-
↳ intensity forecast thresholds.
"""
def determine_governor(config, forecast):
    ci_limit = int(config['carbon_intensity_limit'])

    if forecast <= (0.25 * ci_limit):
        # Less than or equal to 25%
        print("CI forecast less than or equal to 25% of
        ↳ specified limit...")
        return get_governor_weight("performance")
    elif forecast <= (0.5 * ci_limit):
        # Less than or equal to 50%
        print("CI forecast less than or equal to 50% of
        ↳ specified limit...")
        return get_governor_weight("ondemand")
    elif forecast <= (0.75 * ci_limit):
        # Less than or equal to 75%

```

```

    print("CI forecast less than or equal to 75% of
    ↪ specified limit...")
    return get_governor_weight("conservative")
elif forecast <= ci_limit:
    # Greater than forecast
    print("CI forecast within 25% or equal to specified
    ↪ limit...")
    return get_governor_weight("powersave")
elif forecast > ci_limit:
    print("CI forecast exceeds specified limit...")
    return get_governor_weight("powersave")

"""
Start Green Lantern background process. Location and carbon-
↪ intensity forecast are obtained according
to the poll interval set in the config and governor adjusted
↪ accordingly.
"""
def start(config):
    location = get_location(config)

    while True:
        ci_request = get_CI(location, config)
        ci_forecast = ci_request['data'][0]['data'][0]['
        ↪ intensity']['forecast']
        gov = determine_governor(config, ci_forecast)

        print("Carbon intensity forecast: " + str(ci_forecast)
        ↪ )

        try:
            set_governor(gov)
            print("CPU governor: " + get_governor_name(gov))
            print("=====")

        except:
            print("Unable to adjust CPU governor. Please
            ↪ ensure you are using a compatible CPU.")

    time.sleep(config['poll_interval'])

```

```

"""
Start Green Lantern background process. Mock data used in
    ↪ place of real-world data.
Governor adjusted accordingly at interval set by user in the
    ↪ config.
"""
def start_benchmark(config):
    while True:
        ci_forecast = get_mock_CI()
        gov = determine_governor(config, ci_forecast)

        print("Carbon intensity forecast: " + str(ci_forecast)
              ↪ )

        try:
            set_governor(gov)
            print("CPU governor: " + get_governor_name(gov))
            print("=====")

        except:
            print("Unable to adjust CPU governor. Please
                  ↪ ensure you are using a compatible CPU.")

        time.sleep(10)

```

A.3 config.toml

```

[config]
mode = "regional"
location = "GL52"
carbon_intensity_limit = "350"
poll_interval = 10

```

A.4 Calculix Benchmark

The Calculix example programs launched in the benchmarking used for this dissertation can be found at [here](#).

```

for i in $(seq 1 100)

```

```
do
    # blisk
    cd ./blisk
    python3 run_blisk_py.py

    # 2D plate with hole
    cd ../plate_with_hole_2D
    python3 plate_with_hole_2D_py.py

    # 3D plate with hole
    cd ../plate_with_hole_3D
    python3 plate_with_hole_3D_py.py

    cd ../

done
```

A.5 Libquantum Benchmark

Shor's algorithm example implementation in libquantum can be found [here](#).

```
sudo ./jouleit.sh ./shor 4000
```
