

Integrated Energy Measurement Approach on HPC Systems

Master Thesis

University of Basel
Faculty of Science
Department of Mathematics and Computer Science
High Performance Computing Group

Advisor and Examiner: Prof. Dr. Florina Ciorba
Supervisor: Thomas Jakobsche

Author: Hari Narayanan
Email: hari.narayanan@stud.unibas.ch

2.April 2024



Acknowledgement

I would like to express my deepest gratitude to all those who have contributed to the completion of this master's thesis. First and foremost, I am immensely grateful to my examiner, Prof. Dr. Florina Ciorba for giving me the opportunity to do my Masters thesis in her group on such an interesting topic. I would also like to thank Thomas Jakobsche for his invaluable guidance, support, and encouragement throughout this research journey. Their expertise, patience, and constructive feedback have been instrumental in shaping this thesis. I would also like to thank Martin Jacquot and Dr. Iñaki Martínez de Ilarduya from the sciCORE team for all the support in installing requested softwares on the sciCORE cluster and for the support provided to collect the data. I would also like to thank Robert Frank, the system administrator of the miniHPC, for the accesses and softwares required on miniHPC.

Abstract

High-Performance Computing (HPC) systems face significant challenges regarding energy consumption and carbon footprint. Inefficient utilization of resources within HPC applications exacerbates these issues. Existing energy monitoring solutions, such as Green Algorithms, Ganglia/Prometheus, SLURM (Simple Linux Utility Resource Management) energy plugin, and PMT (Power Measurement Toolkit), operate independently and lack integration, comparison, and accessibility, hindering comprehensive energy optimization efforts. To address this gap, we propose an integrated energy measurement framework operating on multiple levels (system, node, job, application) and leveraging various software tools (e.g. PMT, SLURM, Ganglia, Prometheus, LIKWID). This framework aims to provide holistic insights into application energy consumption, and facilitating targeted optimization strategies. Experimental evaluations across different configurations (MPI, OpenMP, hybrid) and system setups, including variations in CPU frequencies, were conducted to validate the proposed solution. Key findings include the correlation between increased ranks/threads and reduced energy consumption, the impact of NUMA, and the influence of CPU frequency on energy efficiency. This thesis underscores the importance of integrated energy monitoring and optimization in HPC environments to mitigate costs, reduce carbon footprint, and promote sustainability. Additionally, it highlights challenges in software installation and usage, which must be addressed for effective implementation of energy-efficient practices in HPC.

Contents

1	Introduction	3
2	Background	7
2.1	High-Performance Computing (HPC) Architecture	7
2.1.1	miniHPC Cluster	7
2.1.2	sciCORE Cluster	7
2.1.3	Programming Paradigms for HPC	10
3	Related Work	12
3.1	Tools and framework that manage energy	15
3.2	Green Algorithms	17
4	Methods	19
4.1	NAS Benchmarks	19
4.2	RAPL vs IPMI	20
4.3	Energy Measurement Tools	21
4.3.1	PMT (Power Measurement Tool)	21
4.3.2	LIKWID (Like I Know What I'm Doing)	23
4.3.3	SLURM Energy Monitoring	26
4.3.4	Prometheus	29
4.3.5	Ganglia	32
4.4	Green Algorithms Code for System Wide Carbon Footprint Generation	35
4.5	Experimental Setup	38
5	Results	40
5.1	OpenMP Experiments	40
5.1.1	miniHPC and sciCORE	40
5.2	MPI Experiments	41
5.2.1	miniHPC	41
5.2.2	sciCORE	41
5.2.3	MPI experiments with 2 nodes	41
5.3	Hybrid Experiments	47
5.4	CPU Frequency vs Energy	49
5.5	Integration of all Energy components: The Utility Tool	49

6	Discussion	52
6.1	OpenMP	52
6.2	MPI	52
6.3	Hybrid	53
6.4	Affect of different CPU frequencies	54
6.5	Corner Cases: NUMA effect	54
6.6	My carbon footprint	55
7	Conclusion	58
7.1	Contributions from the work	58
7.2	Future Work	58
	7.2.1 Job Co-location Strategies	59
8	Appendix	64

Chapter 1

Introduction

Problem Statement The fragmented state of current energy measurement solutions in High-Performance Computing (HPC) systems hinders comprehensive insights into energy consumption patterns. This inhibits the development of targeted optimization strategies for enhancing energy efficiency and sustainability across node, job, and application levels. Overcoming challenges related to programming paradigms, software installation, and utilization is critical for effective implementation of energy-efficient practices in HPC.

Motivation

- (a) **Increasing Operational Costs:** As the computational demands of HPC systems continue to grow, so do their operational costs, especially in terms of energy consumption. Energy-aware systems help in optimizing energy usage, thereby reducing operational expenses. By implementing energy measurement approaches, organizations can accurately track and analyze energy usage patterns to identify areas for improvement and cost savings.
- (b) **Environmental Impact:** HPC systems consume substantial energy, contributing to their environmental footprint. Energy-aware systems and integrated energy measurement approaches allow for better power consumption management, reducing carbon emissions and environmental impact. By optimizing energy usage, HPC facilities can work towards achieving sustainability goals and minimizing their overall environmental footprint.
- (c) **Power Density Challenges:** HPC systems often face power density challenges due to the high concentration of computing resources within limited physical spaces. Energy-aware systems help efficiently manage power distribution and cooling requirements, mitigating power density challenges. Integrated energy measurement approaches provide insights into power usage at various levels of the system architecture, enabling better optimization strategies to effectively address power density issues.
- (d) **Scalability:** In HPC environments, organizations constantly seek to expand computational capabilities to meet increasing demands, making scalability crucial. Energy-efficient systems ensure that organizations achieve scalability without compromising energy efficiency. Integrated energy measurement approaches enable scalability assessments by providing real-time

data on energy consumption, facilitating informed decisions regarding system expansion and resource allocation.

- (e) **Innovation:** Innovation in HPC often revolves around improving performance while reducing energy consumption. Energy-aware systems foster innovation by encouraging the development of more efficient hardware architectures, software algorithms, and system management techniques. Integrated energy measurement approaches support innovation by providing valuable insights into energy usage patterns and guiding the design and implementation of novel energy-saving solutions.
- (f) **Sustainable Objectives:** Sustainability has become critical for organizations across various industries, including HPC. Energy-aware systems align with sustainable objectives by promoting efficient resource utilization and reducing wastage. Integrated energy measurement approaches help monitor progress toward sustainability goals by tracking energy consumption, identifying areas for optimization, and facilitating continuous improvement initiatives.

Existing Solutions In Chapter 3 of the related work, various existing solutions have been thoroughly discussed. These solutions, such as PMT (Power Measurement Toolkit) and LIKWID (Like I Know What I'm Doing), are instrumental in calculating the energy consumption of applications within High-Performance Computing (HPC) environments. Additionally, SLURM (Simple Linux Utility Resource Management) has been identified as a tool capable of estimating the energy usage of jobs based on job accounting data. Furthermore, energy management tools like Ganglia and Prometheus operate at the hardware level, offering insights into energy usage patterns. Despite the availability of these tools, there remains a significant gap in integrating them into a cohesive and holistic solution within the HPC environment. Currently, no comprehensive platform allows system administrators or users to gain a unified overview of energy consumption and identify areas of energy wastage. The current lack of integration is hindering the effective management of energy resources and optimization of efficiency within HPC infrastructures. In essence, while individual tools exist for specific aspects of energy management in HPC, there is a pressing need for a unified solution that seamlessly integrates these tools. Such a holistic platform would empower stakeholders to make informed decisions regarding energy consumption, ultimately leading to enhanced resource utilization and sustainability within HPC environments.

Research Goals Energy consumption of HPC systems is an increasingly important topic to study in light of global concern for sustainability and reducing carbon emissions. The goals of this master thesis are:

- to develop an integrated approach for energy measurement on multiple levels (system, node, job, application).
- to run experiments, extract insights and explore ways with different energy configurations.
- and to report these energy measurements to system administrators and end users.

This research goes beyond analysis and delves into implementation, with a specific focus on replicating the proposed energy measurement framework within the sciCORE HPC infrastructure at the University of Basel. By integrating energy measurement capabilities across node, job, and application levels, a comprehensive understanding of energy utilization within sciCORE's HPC clusters can be achieved.

Proposed Solution

1. Review of Existing Systems:

- The study will critically analyze existing energy-monitoring platforms to discern their capabilities and limitations.
- The existing system where experiments were conducted on sciCORE and miniHPC cluster.

2. Multi-level Energy Measurement:

- The research targets four distinct levels:
 - (a) **Node-level:** Tools like **Ganglia** and **Prometheus** are used to calculate the energy consumption at the node level.
 - (b) **Job-level:** **SLURM**, a job scheduler, will be used, though it has limitations like the inability to measure energy for multiple concurrent jobs on shared assets distinctively.
 - (c) **Application-level:** At the application level, the intricacies of software execution significantly influence energy consumption profiles. Leveraging tools such as **LIK-WID** and **Performance Monitoring Tools (PMT)**, this thesis aims to analyze the energy footprint of individual applications running within HPC environments. By correlating application characteristics with energy consumption data, insights can be gained into optimization opportunities that balance performance with energy efficiency.
 - (d) **System Wide:** Carbon footprint calculator is used to generate system wide carbon footprint for the user.

3. Comparative Analysis:

- Various experiments were set up, altering node configurations, job setups, and application structures and different programming paradigms (**OpenMP**, **MPI** and **Open+MPI**). The outcomes were utilized to compare the energy-monitoring capabilities across the three aforementioned levels and different frequency levels.

4. Unified Data Model:

- Finally a wrapper script has been created that integrates all the energy components and gives the user energy consumption at different levels (node, job and application).

Highlights Main Highlights of the thesis are:

- The experiments were run on two clusters miniHPC and sciCORE.
- Identified and analysed the energy consumption within different programming paradigms like OpenMP, MPI, Hybrid (OpenMP+MPI) on different CPU frequency levels
- Extracted energy at node level, job level and application level and finally generated the carbon footprint for our experiments.
- Integrating energy measurement capabilities at the node, job, and application levels and providing a unified command-line tool aims to facilitate informed decision-making for sustainable and efficient HPC operations.

Outline The document is structured as follows: It begins with an Introduction in Chapter 1, which outlines the study’s goals. Chapter 2 provides background information, giving context to ongoing experiments in related systems, i.e., miniHPC and sciCORE. Moving forward, Chapter 3 delves into existing research and literature on energy use in high-performance computing (HPC) systems. In Chapter 4, the Methods section explains the tools used in the experiments and how they were set up. Results are then presented in Chapter 5, followed by a discussion and analysis of those results in Chapter 6. Finally, Chapter 7 wraps up the thesis by summarizing the main findings, highlighting the study’s contributions, and suggesting directions for future research.

Chapter 2

Background

2.1 High-Performance Computing (HPC) Architecture

High-Performance Computing (HPC) systems are pivotal in addressing complex computational problems across various scientific and engineering domains. These systems typically consist of interconnected clusters of computing nodes designed to deliver massive computational power. In this study, attention has been directed towards two High-Performance Computing (HPC) clusters: miniHPC and sciCORE.

2.1.1 miniHPC Cluster

miniHPC is a distributed computing platform comprised of multiple computing nodes interconnected through high-speed networks at the University of Basel with the HPC research group[1]. miniHPC boasts a peak performance of 28.9 double-precision TFLOP/s. It comprises four types of nodes, as described in Table 2.1. Each node in the miniHPC cluster is equipped with multi-core processors, high-speed memory, and access to shared storage resources. The architecture of miniHPC is designed to facilitate parallel processing and efficient data exchange among nodes. We have primarily used the nodes **cl-node002** and **cl-node003** for our experiments on the partition **xeon**. The architecture/ thread topology is mentioned in the Table 2.2

2.1.2 sciCORE Cluster

The sciCORE facility [4], managed by the scientific computing center at the University of Basel, offers a specialized high-performance computing (HPC) environment tailored to support data and computation intensive research endeavors. sciCORE aims to provide seamless access to HPC resources and expertise, enabling researchers to effectively utilize computational methods in their scientific endeavors, with a focus on facilitating innovative research.

sciCORE presently manages a cutting-edge high-performance computing setup, segmented into three specialized environments catering to distinct scientific requisites. This infrastructure comprises 228 nodes interconnected via InfiniBand and Ethernet 100G, housing around 13,500 CPU cores and offering 70 TB of distributed memory with a substantial disk storage capacity of 11 PB. Currently, the facility accommodates over 800 users, with an annual consumption rate of nearly 30

Table 2.1: Architecture of miniHPC and sciCORE Clusters

Component	Details
miniHPC Cluster	
Nodes	28
Total Cores	166
Total RAM	3392 GB
Total GPUs	2
Total Disk	37 TB
Interconnect	Eth 10G, OmniPath 100G
Operating System	CentOS
sciCORE Cluster	
Nodes	223
Total Cores	14528
Total RAM	79 TB
Total GPUs	128
Total Disk	16 PB
Interconnect	Eth 100G, Infiniband
Operating System	CentOS

Table 2.2: System Configuration for node002 and node003 on partition xeon

CPU	
CPU name	Intel Xeon CPU E5-2640 v4 @ 2.40GHz
CPU type	Intel Xeon Broadwell EN/EP/EX processor
CPU stepping	1
Threads per core	1
Hardware Thread Topology	
Sockets	2
Cores per socket	10
Threads per core	1
Cache Topology	
Level 1	Size: 32 kB Cache groups: (0) to (19)
Level 2	Size: 256 kB Cache groups: (0) to (19)
Level 3	Size: 25 MB Cache groups: (0-9) and (10-19)
NUMA Topology	
NUMA domains	2
Domain 0	Processors: (0-9) Distances: 10 (to same domain), 21 (to other domain) Free memory: 31268.9 MB Total memory: 31914 MB
Domain 1	Processors: (10-19) Distances: 21 (to same domain), 10 (to other domain) Free memory: 31573.4 MB Total memory: 32253.8 MB

million CPU hours, encompassing 14 million job executions. The technical details of the cluster are mentioned in the table 2.1. We have used the nodes sgi63 and sgi64 on partition a100 to extract the energy components for our experiments. The system configuration for the nodes is mentioned in the table 2.3.

Table 2.3: System Configuration of sgi63 and sgi64 on partition a100

CPU	
CPU name	AMD EPYC 7742 64-Core Processor
CPU type	AMD K17 (Zen2) architecture
CPU stepping	0
Threads per core	1
Hardware Thread Topology	
Sockets	2
Cores per socket	64
Threads per core	1
Cache Topology	
Level 1	Size: 32 kB Cache groups: (0) to (127)
Level 2	Size: 512 kB Cache groups: (0) to (127)
Level 3	Size: 16 MB Cache groups: (0-3), (4-7), ..., (124-127)
NUMA Topology	
NUMA domains	2
Domain 0	Processors: (0-63) Distances: 10 (to same domain), 32 (to other domain) Free memory: 404396 MB Total memory: 515716 MB
Domain 1	Processors: (64-127) Distances: 32 (to same domain), 10 (to other domain) Free memory: 500014 MB Total memory: 516066 MB

2.1.3 Programming Paradigms for HPC

Efficient utilization of HPC resources relies heavily on employing suitable programming paradigms that leverage parallelism and optimize computational workflows. In this study, we investigated the energy consumption over three key programming paradigms commonly used in HPC environments:

1. **OpenMP (Open Multi-Processing):** OpenMP is a widely adopted shared-memory parallelization model that allows developers to parallelize code across multiple threads within a single node. It provides a simple and flexible approach for leveraging parallelism in HPC applications, particularly suitable for shared-memory architectures found in miniHPC and sciCORE nodes.
2. **MPI (Message Passing Interface):** MPI is a message-passing library standard used for parallel computing across distributed memory architectures. It enables communication and coordination between separate nodes in an HPC cluster, making it well-suited for scaling applications across multiple computing resources. Both miniHPC and sciCORE support MPI, facilitating distributed computing tasks across their respective clusters.
3. **Hybrid Programming:** Hybrid programming combines the strengths of both OpenMP and MPI paradigms to achieve optimal performance in HPC applications. By leveraging shared-memory parallelism within individual nodes (using OpenMP) and distributed-memory parallelism across nodes (using MPI), hybrid programming offers a powerful approach to harnessing the computational capabilities of modern HPC systems.

This thesis scrutinizes and contrasts the energy consumption profiles of distinct programming paradigms (OpenMP, MPI, and hybrid) on miniHPC and sciCORE platforms. By analyzing the energy characteristics of these paradigms, we have attempted to furnish insights into their efficiency and suitability for diverse computational tasks in HPC environments. Furthermore, this study seeks to enrich the broader comprehension of energy-conscious programming methodologies within high-performance computing.

Chapter 3

Related Work

This chapter presents a critical analysis of the related literature and highlights potential areas for improvement.

Ghislain Landry et al. [10] proposed a runtime framework for improving the energy efficiency of HPC systems without requiring prior knowledge of the applications running on the system. The framework consists of two main components: A phase detection method identifies the different phases of an HPC system's workload. A phase is when the system's workload exhibits similar characteristics. The phase detection method uses statistical and machine learning techniques to identify the different phases. A partial phase recognition technique uses the results of the phase detection method to adjust the system's power and performance settings dynamically. The technique considers the current phase of the workload, the system's resources, and the user's performance requirements. The paper evaluates the effectiveness of the proposed framework using real-world workloads and benchmarks. The results show that the framework can save up to 15 percent of energy without degrading the performance of the applications.

Jumie Yuventi and Roshan Mehdizadeh [44] discussed the shortcomings of PUE (Power Usage Effectiveness). They propose several alternative metrics for communicating data center energy consumption. These metrics include data center infrastructure efficiency (DCIE), energy use intensity (EUI), and carbon footprint. The paper argues that these alternative metrics are more accurate and informative than PUE and should be used to communicate data center energy consumption.

The paper by Axel Auweter et al. [5] evaluates the energy-aware scheduling capabilities of the IBM LoadLeveler on SuperMUC, a leading HPC system. The study examines how LoadLeveler predicts execution times and power consumption based on different CPU frequencies, contrasting these predictions with accurate benchmark measurements. Given that the prediction model aligns well with actual application workloads, it evaluated a substantial part of SuperMUC's applications. The research paper proposes a two-step approach to energy-aware scheduling on heterogeneous supercomputers. The first step is to predict the power consumption of each application at different CPU frequencies. The second step is to select the CPU frequencies for each application, considering the power and performance predictions and the application deadlines. The paper evaluates the approach using a set of real-world applications and shows that it can achieve significant energy savings without significantly impacting performance.

Tsafack Chetsa et al. [11] proposed a method for predicting and improving the energy performance of HPC systems. The method uses hardware performance counters to collect data about

the system’s behavior. This data is then used to train a model to predict the system’s energy consumption. The model can then optimize the system’s performance by adjusting the CPU frequency or the number of cores used. The paper evaluates the proposed method using a set of real-world HPC applications and shows that it can achieve significant energy savings without significantly impacting performance.

”Power Signatures of High-Performance Computing Workloads” by Jacob Combs et al. [12], shows that simple feature-based signatures can accurately capture the power consumption behavior of HPC workloads. They used a dataset of 13 HPC workloads running on four different hardware platforms to train a classifier that could distinguish between the workloads based on their power signatures, where classifiers achieved an accuracy of 85 percent.

The paper ”Predicting the Energy and Power Consumption of Strong and Weak Scaling HPC Applications” by Hayk Shoukourian et al.[37] presents a method for predicting HPC applications’ energy and power consumption with strong and weak scaling. The proposed method uses a two-step approach: first, a power model is used to predict the power consumption of each application at a single processor; second, a scaling model is used to predict the application’s power consumption at multiple processors. The research paper examines the effectiveness of the proposed method by testing it on a range of HPC applications. It accurately forecasts energy and power consumption of HPC applications with strong and weak scaling.

The paper by Christian Conficoni et al.[13] proposes a control strategy to achieve significant energy savings for hot-water-cooled supercomputers. The strategy uses a model-based approach to predict the cooling system’s power consumption. Then, it optimizes the operation of the cooling system, such as by adjusting the pump speed and the water temperature. The paper evaluates the proposed strategy using a simulation model and shows that it can achieve energy savings without significantly impacting the supercomputer’s performance.

A global survey [22] of HPC centers found that most use energy- and power-aware job scheduling and resource management solutions. The most common approaches are to throttle the CPU frequency of idle nodes, power them down, use DVFS, and use workload-aware scheduling algorithms. There is a growing interest in using more sophisticated energy- and power-aware job scheduling and resource management solutions. However, there are still some challenges in this area, such as the need for more accurate power models for HPC systems, the difficulty of incorporating energy and power constraints into job scheduling algorithms, and the need to balance energy efficiency with performance.

Connor Imes et al. [20] proposed a method for using machine learning classifiers to improve the energy efficiency of application resource scheduling in HPC systems. The proposed method is a promising approach that can achieve significant energy savings without significantly impacting the performance of the applications. However, some challenges must be addressed before the method can be widely adopted, such as dealing with the uncertainty of application resource requirements and system power consumption and developing more efficient algorithms for scheduling applications using the predicted resource settings.

The paper by Jan Weglarz et al.[42] discusses APIs and tools for controlling energy and power consumption in high-performance computing (HPC) systems. It analyzes the approaches, control methods, optimization criteria, programming examples, and benchmarks used in state-of-the-art research on energy-aware HPC. The paper also presents solutions for specific types of HPC systems, such as workstations, clusters, grids, and clouds. It discusses optimization metrics, such as execution time, energy used, power consumption, and temperature. The paper also describes the control methods used in existing approaches, such as scheduling, DVFS/DFS/DCT, power cap-

ping, application optimizations, and hybrid approaches. Finally, the paper presents open areas and recommendations for future research.

AI-Driven Holistic Approach to Energy Efficient HPC [40] proposes a new approach to energy efficiency in high-performance computing (HPC) systems. The approach uses artificial intelligence (AI) to model the performance and power consumption of HPC systems and to develop scheduling policies that optimize energy efficiency while meeting the performance requirements of applications. The proposed approach takes a holistic view of energy efficiency, considering the entire HPC system, from the hardware to the software. It uses AI to model the performance and power consumption of all the components in the system, which allows it to develop scheduling policies that optimize energy efficiency while meeting the performance requirements of applications.

The paper by Gence Ozar et al. [32] introduces a method for constructing a predictive energy model tailored for High-Performance Computing (HPC) runtime systems, leveraging supervised learning capabilities. Initially, the process involves gathering sensor data and relevant performance metrics directly from the HPC system. This data then serves as the foundation upon which a supervised learning algorithm is trained, with the specific intent of forecasting the CPU's power consumption and the anticipated number of instructions that will be retired. Armed with this predictive model, the system can astutely determine the most suitable CPU frequency during a job's execution, aiming to optimize energy efficiency and minimize overall consumption within the HPC system.

Laurens Vershuis et al. [41] proposed a method for holistic analysis of data center operations, using fine-grained data to reveal new operational aspects and help improve data center performance and efficiency. The authors argue that a holistic approach is needed that considers the entire system, from the hardware to the software. The authors showcase the advantages of their method by implementing it on a data center infrastructure consisting of more than 300 nodes. They made over 30 significant observations that can aid in tasks related to performance engineering, such as forecasting data center load and designing data center infrastructure.

Eishi Arima et al. [3] have introduced the convergence of malleability and power management in HPC systems. Malleability refers to the ability of an HPC system to dynamically scale its resources to meet the changing needs of applications. In contrast, power management refers to the techniques used to reduce the power consumption of an HPC system. The paper proposes a framework for converging malleability and power management in HPC systems based on dynamic resource allocation, co-scheduling, and application malleability. The paper evaluates the framework using a simulation study and shows that it can achieve significant energy savings over traditional approaches.

The paper by Issa Saba et al. [36] proposes a new approach to energy-efficient scheduling of applications on CPU-GPU heterogeneous systems. The approach uses machine learning to optimize co-scheduling, resource partitioning, and power capping jointly. First, it uses machine learning to predict the execution time of each application on each type of resource. Then, it uses this information to co-schedule applications to minimize the overall execution time of the applications. It also uses machine learning to partition resources between applications to minimize contention and maximize performance. Finally, it uses machine learning to cap the system's power consumption to meet power constraints. The paper evaluates the proposed approach using a simulation study and shows that it can achieve significant energy savings over traditional approaches.

In the research paper by Thomas Jakobsche et al. [39], they introduce a research methodology that examines approximately 350,000 jobs executed over a two-month span. The goal was to uncover the relationship between the causes of job delays and potential strategies to lessen these wait times.

The results of these analyses could aid both users and system operators in understanding the factors behind prolonged job wait times and methods to curtail them. The primary cause identified was that these delays stem from Quality of Service (QoS) limitations.

One of the studies by Pawel Bratek et al. [8] investigates the benefits of heterogeneous Dynamic Voltage and Frequency Scaling (DVFS) in improving the energy efficiency of data-parallel applications on multicore CPU systems. Using two use cases, it demonstrated up to 20 percent energy savings with heterogeneous scaling on a specific server. To facilitate this, two efficient pruning algorithms were introduced, significantly reducing the computational cost compared to the brute-force method. The approach also proved effective on 64-core AMD EPYC processors, achieving up to a 13 percent efficiency advantage over traditional DVFS methods.

3.1 Tools and framework that manage energy

Matthias Maiterth et al. [29] have introduced GEOPM, a framework for exploring power and energy optimizations on heterogeneous platforms. GEOPM is an open-source and collaborative framework that fosters community research on power management strategies. Researchers can share their work and collaborate on new ideas through the platform. GEOPM is also designed to be portable and scalable for various hardware platforms. The paper presents a case study of how GEOPM was used to develop a power management strategy for the SuperMUC supercomputer. The case study shows how GEOPM can be used to optimize the power consumption of a supercomputer while still meeting the performance requirements of the applications running on the system. The paper outlines the challenges and opportunities for future research on GEOPM. Some of the significant challenges include the need for more precise power models for heterogeneous platforms, more efficient optimization algorithms for power management, and more realistic simulation models of power management strategies.

Neha Gholkar et al. [16] proposed a method called PShiter that can improve the performance of HPC jobs by dynamically shifting power. PShiter uses a feedback controller to monitor the performance of the job and adjust the power allocation accordingly. The controller can detect when a job is experiencing performance degradation due to power constraints and take corrective action to improve the performance. The paper evaluates the proposed method using an HPC system simulation model and shows that PShiter can significantly improve performance for various HPC jobs. It discusses challenges and opportunities for future research in dynamic power shifting for HPC jobs. These challenges include developing more accurate power models for HPC systems, developing more efficient feedback controllers, and addressing the challenges of coordinating power shifting across multiple jobs. The paper also discusses some of the opportunities for future research in this area, such as developing methods for incorporating uncertainty into the power models, developing methods for optimizing the performance-power trade-off for HPC jobs, and developing methods for integrating dynamic power shifting with other techniques for improving the performance of HPC jobs.

The paper "COUNTDOWN Slack: a Run-time Library to Reduce Energy Footprint in Large-scale MPI Applications" by Daniele Cesarini et al. [9] proposes a run-time library called COUNTDOWN Slack that can reduce the energy footprint of large-scale MPI applications. COUNTDOWN Slack works by exploiting Slack in the communication phases of MPI applications. Slack is the time difference between the end of a communication phase and the start of the next computation phase. COUNTDOWN Slack identifies Slack and then uses it to reduce the system's power consumption. The paper evaluates COUNTDOWN Slack using a variety of MPI applications and shows that

COUNTDOWN Slack can achieve significant energy savings without significantly impacting the performance of the applications.

EAR[14] is an energy management framework for supercomputers developed by the Barcelona Supercomputing Center (BSC). EAR is designed to optimize the energy efficiency of supercomputers by dynamically managing the power consumption of the system’s nodes. EAR works by monitoring the power consumption of the nodes and the workload running on them. It then uses this information to determine how to allocate resources to the nodes to maximize energy efficiency. EAR can also be used to implement various energy-saving policies, such as throttling the CPU frequency of idle nodes or powering down nodes that are not being used. EAR has been evaluated on various supercomputers and has been shown to achieve significant energy savings without significantly impacting the performance of the applications running on the system.

The Power Measurement Toolkit (PMT) [15] is a high-level software library for collecting power consumption measurements on various hardware platforms. It is written in C++ and is available for Linux-based systems. PMT provides a standard interface to measure the energy usage of devices such as CPUs and GPUs in critical application sections. It can be used to monitor and evaluate applications’ energy efficiency and optimize the power consumption of HPC systems. PMT is easy to use, provides accurate power measurements, and is extensible and open-source.

Researchers [4] integrated the Power Measurement Toolkit (PMT) into the SPH-EXA simulation framework. This integration reveals energy consumption patterns across multiple devices, aiding in identifying areas of the simulation code that could be more energy-efficient. Using subsonic turbulence simulations for validation, they compared PMT’s energy data with Slurm system readings. They also examined the energy usage in systems with adjustable GPU frequencies, highlighting the potential for energy savings by optimizing GPU compute frequencies. It is noted, however, that current GPUs lack the nuanced frequency scaling of CPUs and do not allow users to control their dynamic frequency scaling, limiting energy conservation opportunities.

Table 3.1: Comparative Analysis of EAR, PMT, and PShifter Systems

Feature	EAR	PMT	PShifter
Purpose	Reduces energy consumption of computing systems	Provides tools for managing power consumption	Improves performance of HPC jobs
Techniques used	Dynamic power management, workload management	Dynamic power management, workload management, energy accounting	Dynamic power shifting
Flexibility	Flexible	Flexible, scalable	Not as flexible
Scalability	Scalable	Scalable	Not as scalable
Energy awareness	Energy-aware	Not as energy-aware	Not as energy-aware

3.2 Green Algorithms

The **GREENER** principles provided by Loïc Lannelongue et al.[25] serve as a comprehensive guide to promote environmentally conscious computational science. The six cornerstone principles are:

- **Governance:** Advocate for strong governance frameworks to underline the importance of green computational science.
- **Responsibility:** Ensure entities are answerable for the environmental impact of their computational procedures.
- **Estimation:** Utilize techniques to quantify and relay the environmental effects of computational processes.
- **Energy and embodied impacts:** Examine comprehensive lifecycle influences within computational endeavors, spanning hardware to software.
- **New collaborations:** Forge alliances among computational scholars, environmental advocates, and decision-makers to tackle the sustainability issues inherent in computational science.
- **Education and Research:** Bolster educational and research initiatives concentrating on the environmental aspects of computational science to unearth and promote sustainable practices.

By staying true to the **GREENER** principles, all involved parties can significantly mitigate the ecological impact of computational science, championing the larger cause of environmental preservation in the face of climate change.

Another paper by Loïc Lannelongue et al.[28] addresses the increasing concern regarding the carbon footprint of computational research, attributed to factors such as the energy consumption of computers, embodied emissions of hardware, and data storage and transportation emissions. To estimate this footprint, the authors introduce several methodologies, including the LEAF (Laboratory Efficiency Assessment Framework) framework. Emphasizing the escalating energy demands of computers, the paper underscores the significance of evaluating and understanding this footprint to steer computational science towards a more environmentally sustainable path.

Loïc Lannelongue, Michael Inouye, and collaborators highlight [17] s the bioinformatics field’s substantial and growing carbon footprint, primarily attributed to computer energy consumption, hardware emissions, data storage and transport. To mitigate this, the authors suggest strategies like utilizing renewable energy sources, adopting energy-efficient computing infrastructure, algorithm optimization,[27] and facilitating data sharing. These measures, they argue, are essential to ensure sustainability in bioinformatics.

In the article ”Carbon footprint, the (not so) hidden cost of high-performance computing” [24], multiple strategies exist for curtailing the carbon footprint of High-Performance Computing (HPC):

- Transitioning to more energy-conscious computing architectures.
- Refining algorithms to lessen computational demands.
- Promoting data-sharing practices, thereby reducing storage and transportation overheads.
- Harnessing renewable energy sources for HPC infrastructure.

Innovations, like quantum computing[26], introduce potential avenues for further reductions. Quantum systems, known for their energy frugality compared to traditional setups, may soon take over certain HPC functionalities.

Tackling the carbon implications of HPC is multifaceted, but with a variety of available strategies, we can step forward in our battle against climate change. In this thesis, the Green Computing aspect is bound to be very important because the idea is to have sustainable methods that we continue using HPC facilities while putting less pressure on the environment. We need to balance performance and sustainability, which would lead to further innovations.

Chapter 4

Methods

4.1 NAS Benchmarks

NAS (NASA Advanced Supercomputing) Benchmarks [6] are a suite of benchmarks developed by NASA to evaluate the performance of parallel supercomputers and high-performance computing (HPC) systems. These benchmarks are designed to stress different aspects of HPC systems, including computation, communication, and memory performance. They are widely used in the HPC community to assess and compare the performance of various hardware architectures, compilers, libraries, and system configurations. Below is a description of the NAS benchmarks:

- A. CG (Conjugate Gradient):** CG benchmark evaluates the performance of sparse matrix-vector multiplication and iterative solution techniques. This solves a system of linear equations using the conjugate gradient method. This benchmark stresses both floating-point computation and memory bandwidth.
- B. MG (Multi-Grid):** MG benchmark assesses the performance of multi-grid methods for solving elliptic partial differential equations. It uses a series of mesh refinements and smoothing operations to solve the equations iteratively. MG benchmark evaluates both computation and memory access patterns.
- C. FT (Fast Fourier Transform):** FT benchmark measures the performance of the fast Fourier transform (FFT) algorithm. It computes the discrete Fourier transform of a 3D complex array. This benchmark primarily evaluates floating-point computation performance.
- D. IS (Integer Sort):** IS benchmark evaluates the performance of sorting algorithms. It sorts a large array of integers using the radix exchange sort algorithm. IS benchmark primarily assesses memory access patterns and cache efficiency.
- E. EP (Embarrassingly Parallel):** EP benchmark measures the performance of simple embarrassingly parallel operations. It computes the dot product of two large vectors. This benchmark primarily evaluates floating-point computation performance and scalability.
- F. BT (Block Tri-diagonal):** BT benchmark evaluates the performance of solving a block tri-diagonal system of equations. It simulates the three-dimensional heat equation in a block-structured grid. BT benchmark stresses both computation and communication performance.

G. LU (Lower-Upper Symmetric Gauss-Seidel): LU benchmark assesses the performance of solving a system of linear equations using the Lower-Upper Symmetric Gauss-Seidel (LU-SGS) method. It simulates the solution of a 3D partial differential equation. LU benchmark evaluates both computation and communication performance.

H. SP (Scalar Pentadiagonal): SP benchmark evaluates the performance of solving a scalar pentadiagonal system of equations. It simulates the solution of a 3D scalar transport equation on a regular grid using a fifth-order accurate finite difference method. The problem involves solving a pentadiagonal system of equations at each grid point. SP benchmark primarily assesses both computation and communication performance, as well as memory access patterns.

These NAS benchmarks provide a standardized set of workloads for evaluating and comparing the performance of HPC systems. They are often used in research, procurement, and optimization efforts to assess the suitability of hardware and software configurations for specific computational workloads. We have used only the SP, BT and LU as these represent Pseudo Applications which are closer to general applications and are a representation of real world applications as show in Figure 4.1.

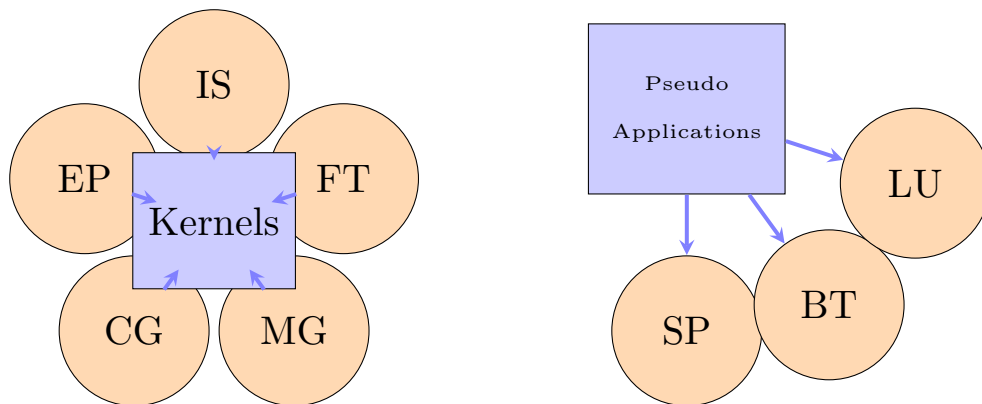


Figure 4.1: NAS Benchmarks

4.2 RAPL vs IPMI

RAPL (Running Average Power Limit) sensors and **IPMI** (Intelligent Platform Management Interface) sensors[21] serve different purposes in computer hardware monitoring. In this thesis, the Power Measurement Toolkit has been used, which extracts the data at the application level to extract the energy information via RAPL sensors. IPMI sensors are used across the whole node, and we have monitored the energy extracted from IPMI via Ganglia (in miniHPC) and Prometheus (in sciCORE). Table 4.1 shows the clear distinction on how the sensors differ from each other.

Aspect	RAPL	IPMI
Scope	Provides energy consumption data for specific components (e.g., CPU, DRAM, cache).	Provides a comprehensive view, covering all components of each node in a system.
Granularity	Fine-grained view at the component level.	Broader perspective across the entire node.
Limitation	May not cover all components, focusing mainly on CPU, memory, and cache.	Offers insights into overall power usage but may lack detailed component-level data.
Advantage	Allows monitoring and analysis of energy consumption at specific hardware elements.	Provides insights into the overall power usage of the hardware.

Table 4.1: Comparison between RAPL and IPMI

4.3 Energy Measurement Tools

4.3.1 PMT (Power Measurement Tool)

The Power Measurement Toolkit (PMT)[15] is a C++ library designed specifically for Linux systems. Its purpose is to interact with the APIs provided by hardware vendors to gather data on power consumption. It supports a range of hardware architectures, including GPUs (using NVML for NVIDIA and rocm-smi for AMD), CPUs (using RAPL or LIKWID), and other architectures like Xilinx FPGAs. Additionally, it can interface with physical power sensors such as PowerSensor2.

PMT’s main functionality involves a background thread within the profiled application that communicates with the chosen backend (such as NVML) to collect power consumption information. The Figure 4.2 is a sample example on how the **Rapl-test** reports energy. The frequency of sampling depends on the hardware and backend being used; for instance, NVML supports sampling up to every 10 milliseconds, while RAPL supports sampling up to every 500 milliseconds.

To install PMT, users can utilize either CMake or a Spack recipe. The library operates in two modes: dump-mode, which records timestamps and power measurements for later analysis, and measurement-mode, which provides the average power consumption of the profiled code for quick energy-efficiency estimations.

Integration into C++ and Python applications involves adding PMT functionalities to the code. For C++, this includes including the PMT header, initializing PMT, and wrapping the region of interest with activation and deactivation function calls. In Python, integration can be simplified using Python bindings or decorators provided by PMT as shown in 4.3

PMT introduces a minimal overhead, typically around 1 millisecond in C++ and 10 milliseconds in Python. However, this overhead accumulates when multiple decorators are employed.

In summary, PMT offers a flexible solution for monitoring power consumption across various hardware architectures, with straightforward integration into both C++ and Python applications. PMT was installed in both miniHPC and sciCORE clusters.


```
[naraya0001@dmi-cl-login ~]$ Rapl-test echo "This is my energy consumption"
This is my energy consumption
Runtime: 0.00669694 s
Joules: 1 J
Watt: 149.322 W
```

Figure 4.2: Sample Rapl-Test with PMT

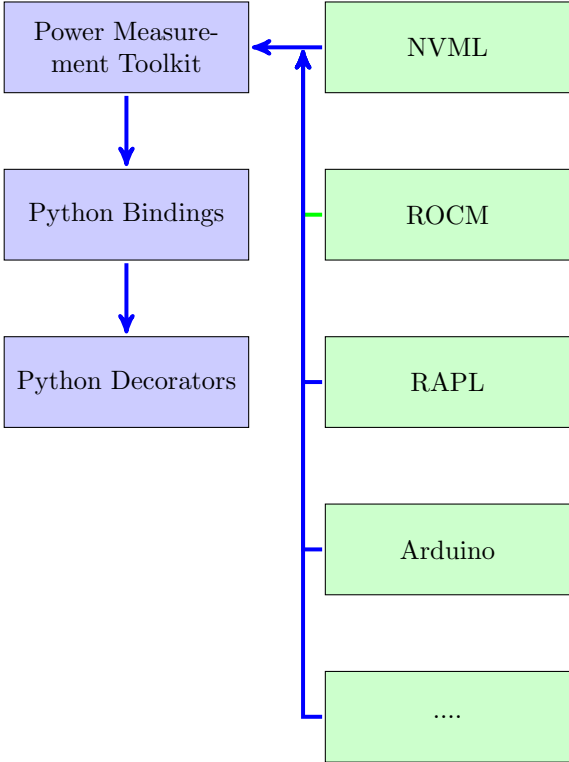


Figure 4.3: Power Measurement Toolkit structure: the library interfaces with different power measurement back ends. The library also includes Python bindings and decorators to ease the measurement for application users.^[15]

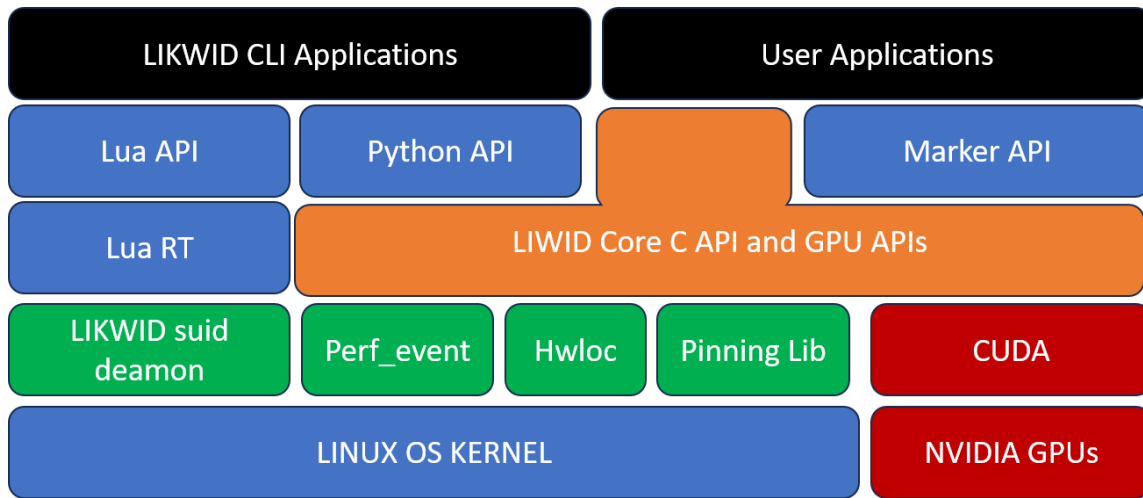


Figure 4.4: LIKWID 5 Architecture [38]

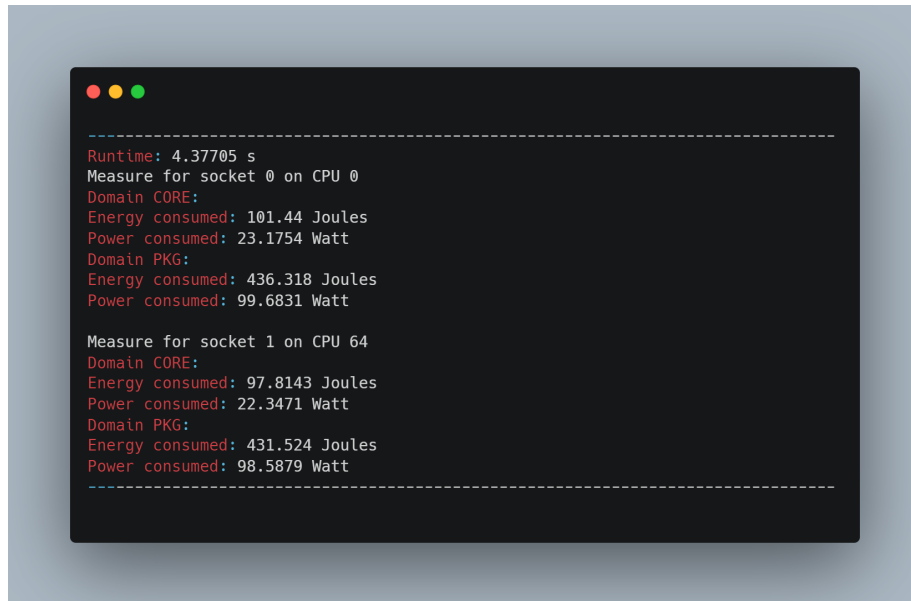
4.3.2 LIKWID (Like I Know What I’m Doing)

LIKWID[19] is a performance analysis tool designed for High-Performance Computing (HPC) systems. It provides a set of command-line utilities and APIs for analyzing the performance of parallel applications running on multi-core processors. The architecture of LIKWID consists of several components as shown in the the Figure 4.4:

- **Data Acquisition Layer (DAL) [18]:**The DAL is responsible for collecting hardware performance counters and other relevant metrics from the underlying hardware. It uses platform-specific interfaces (such as PAPI, perf_events, or APM) to access performance monitoring features provided by the CPU architecture as mentioned in the 4.4 as Linux OS Kernel and NVIDIA GPU’s
- **likwid-suid Daemon [18]:** The likwid-suid daemon is a setuid root utility in LIKWID that allows non-root users to access hardware performance counters. By default, accessing hardware performance counters usually requires root privileges due to security concerns. However, the likwid-suid daemon addresses this limitation by providing a secure mechanism for non-root users to access these counters. This daemon is responsible for managing access to hardware performance counters and ensuring that users have the necessary permissions to collect performance data.
- **HWLOC (Hardware Locality) Library [18]:** HWLOC is an open-source library used to discover and represent the hierarchical topology of modern computing systems. LIKWID utilizes HWLOC to gather information about the system’s hardware topology, including details such as the number of cores, sockets, caches, NUMA (Non-Uniform Memory Access) domains, and their interconnections. This information is crucial for optimizing thread/process placement, memory access patterns, and affinity settings to improve performance in multi-core and parallel computing environments.

- **Pinning Library [18]:** LIKWID provides a pinning library that allows users to bind threads or processes to specific CPU cores. CPU pinning is a technique used to control the placement of threads/processes on CPU cores, which can help improve performance by reducing cache contention, enhancing cache locality, and minimizing NUMA effects. The pinning library in LIKWID facilitates CPU pinning operations, enabling users to optimize thread/process placement for better performance.
- **Perf_event Component [18]:** Perf_event is a Linux kernel subsystem that provides an interface for accessing hardware performance monitoring features. LIKWID utilizes Perf_event to access hardware performance counters and collect performance data. Perf_event allows LIKWID to monitor various aspects of processor performance, including cache misses, branch mispredictions, instruction throughput, and other relevant metrics.
- **Lua API (Application Programming Interface):** LIKWID provides a Lua API that allows users to interact with LIKWID's core functionality programmatically using Lua scripts. With the Lua API, users can access hardware performance counters, collect performance data, perform analyses, and customize monitoring tasks. This API abstracts the low-level details of LIKWID's core library (*likwid-perfctr*), providing a higher-level scripting interface for easier use.
- **Lua RT (Lua Runtime) [18]:** Lua RT refers to the Lua runtime environment provided within LIKWID for executing Lua scripts. It allows users to write scripts in Lua to perform various tasks related to performance monitoring and analysis. Lua RT provides the necessary runtime environment for executing Lua scripts within the context of LIKWID, ensuring compatibility and integration with LIKWID's core functionality.
- **Marker API [18]:** LIKWID provides a Marker API that allows developers to insert instrumentation points (markers) into their code. Markers serve as points of interest where LIKWID can start or stop collecting performance data. Developers can use markers to identify specific sections of code for performance analysis.
- **Counting Groups [18]:** LIKWID organizes performance counters into counting groups. Counting groups represent logical collections of hardware events related to specific performance aspects, such as cache behavior, floating-point operations, or memory accesses. Users can select which counting groups to monitor based on their analysis requirements.
- **Profile API [18]:** LIKWID provides a Profile API that allows users to define and manage performance analysis profiles. Profiles specify the configuration of performance counters and other settings for a particular analysis task. Users can create, save, and load profiles to streamline the performance analysis workflow.
- **Control and User Interface (CUI) [18]:** The CUI provides a command-line interface for interacting with LIKWID. It allows users to specify which performance metrics they are interested in collecting and analyzing. Users can configure LIKWID through command-line options or configuration files.

Overall, the architecture of LIKWID [18] is designed to provide a flexible and extensible framework for collecting, processing, and analyzing performance data on HPC systems. Since PMT only offers the extraction of energy from RAPL sensors (only for Intel), we have used LIKWID, which is more



```
-----  
Runtime: 4.37705 s  
Measure for socket 0 on CPU 0  
Domain CORE:  
Energy consumed: 101.44 Joules  
Power consumed: 23.1754 Watt  
Domain PKG:  
Energy consumed: 436.318 Joules  
Power consumed: 99.6831 Watt  
  
Measure for socket 1 on CPU 64  
Domain CORE:  
Energy consumed: 97.8143 Joules  
Power consumed: 22.3471 Watt  
Domain PKG:  
Energy consumed: 431.524 Joules  
Power consumed: 98.5879 Watt  
-----
```

Figure 4.5: Sample Result of likwid-powermeter command

diverse and supports multiple architectures other than Intel. It offers a range of features to support performance tuning, optimization, and debugging of parallel applications.

LIKWID [35] offers the likwid-powermeter tool to access and measure energy consumption using the RAPL interfaces as shown in the Figure 4.5. Here, you can see the power measurement calculated at the socket level. This tool allows you to:

- **Query energy consumption:** Retrieve energy readings for specific packages (e.g., CPU, memory) within a certain time period.
- **Calculate power consumption:** Based on the energy readings and time interval, calculate the average power consumption in Watts.
- **View Turbo Mode info:** Get information about available Turbo Mode steps on Turbo Mode enabled processors.

likwid-perfctr and Energy Events: While likwid-powermeter offers basic energy information, it is often beneficial to analyze energy consumption alongside other performance metrics. LIKWID's likwid-perfctr tool enables this by providing access to RAPL counters as "events." These events can be included alongside other performance counters when configuring and reading data with likwid-perfctr as shown in the Figure 4.6. This allows for:

- **Correlating energy consumption with other performance metrics:** Analyze how different events or configurations, such as thread affinity or scheduling strategies, impact both performance and energy consumption.
- **Custom analysis and visualization:** Integrate energy data with other performance data for comprehensive analysis and visualization using tools like scripts or external libraries.

Limitations

- **Basic energy measurement:** `likwid-powermeter` provides basic energy and power readings.
- **Requires user configuration:** Selecting appropriate energy events with `likwid-perfctr` requires understanding specific hardware capabilities and desired analysis goals.

4.3.3 SLURM Energy Monitoring

SLURM Energy [43] is an extension of the SLURM workload manager that provides energy-aware scheduling capabilities. It enables users to allocate computing resources based on energy consumption, allowing for more efficient use of power in large-scale computing environments.

SLURM Energy monitoring [2] typically involves the following steps:

1. **Configuration:** SLURM Energy needs to be properly configured on the system to monitor energy consumption. This may involve setting up power measurement tools and integrating them with SLURM.
2. **Job Submission:** Users submit jobs to SLURM with energy monitoring options enabled. This allows SLURM to track energy consumption during job execution.
3. **Monitoring:** During job execution, SLURM monitors energy usage at the node level and possibly at other levels (e.g., job level, application level).
4. **Reporting:** SLURM provides reports on energy consumption for completed jobs. These reports may include total energy usage, energy usage per node, and other relevant metrics.
5. **Resource Allocation:** Based on energy consumption data, SLURM can make informed decisions about resource allocation for future jobs. It may prioritize energy-efficient nodes or adjust job scheduling to minimize overall energy usage.

SLURM Energy monitoring [2] can help organizations optimize their computing infrastructure for energy efficiency, leading to cost savings and reduced environmental impact. SLURM relies on plugins to gather energy consumption data from various sources:

- **Hardware Sensors:** Plugins like ‘Energy Accounting’ and ‘External Sensors’ offer data collection mechanisms.
- **Energy Accounting’ Plugin:** Utilizes in-band interfaces like IPMI (Intelligent Platform Management Interface) or RAPL (Running Average Power Limit) on compatible hardware to access power data directly from compute nodes.
- **External Sensors’ Plugin:** Collects data from external systems like power meters or monitoring tools managed outside SLURM.
- **Data Collection:** These plugins periodically sample energy consumption data (power draw) and temperature (if supported) for each compute node.

```

Group 1: ENERGY
+-----+-----+-----+
|          Event          | Counter | HWThread 2 |
+-----+-----+-----+
| INSTR_RETIRED_ANY      | FIXC0   | 1129008    |
| CPU_CLK_UNHALTED_CORE  | FIXC1   | 1216067    |
| CPU_CLK_UNHALTED_REF   | FIXC2   | 1216320    |
| TEMP_CORE              | TMP0    | 27         |
| PWR_PKG_ENERGY         | PWR0    | 0.1141     |
| PWR_PP0_ENERGY         | PWR1    | 0          |
| PWR_DRAM_ENERGY        | PWR3    | 0.0226     |
+-----+-----+-----+

+-----+-----+-----+
|          Metric          | HWThread 2 |
+-----+-----+-----+
| Runtime (RDTSC) [s]    | 0.0038    |
| Runtime unhaltd [s]    | 0.0005    |
| Clock [MHz]            | 2393.9295 |
| CPI                    | 1.0771    |
| Temperature [C]        | 27        |
| Energy [J]              | 0.1141    |
| Power [W]               | 29.9181   |
| Energy PP0 [J]         | 0         |
| Power PP0 [W]          | 0         |
| Energy DRAM [J]        | 0.0226    |
| Power DRAM [W]         | 5.9268    |
+-----+-----+-----+

```

Figure 4.6: Sample result of **perfctr** in LIKWID

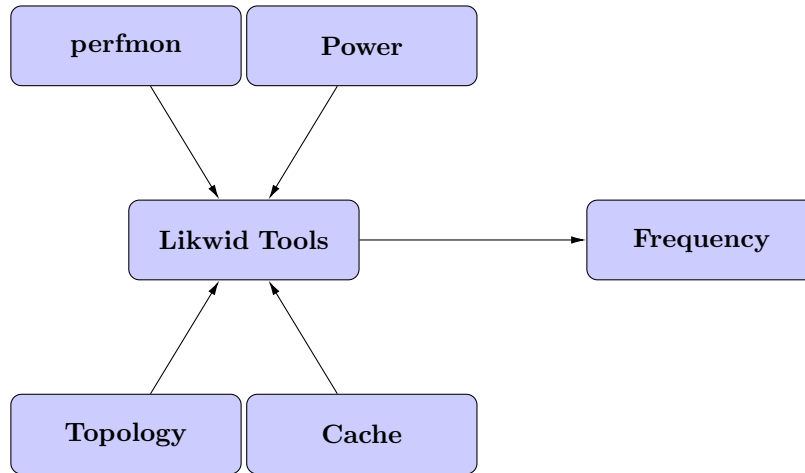


Figure 4.7: General Overview of LIKWID communication

Data Processing and Accounting

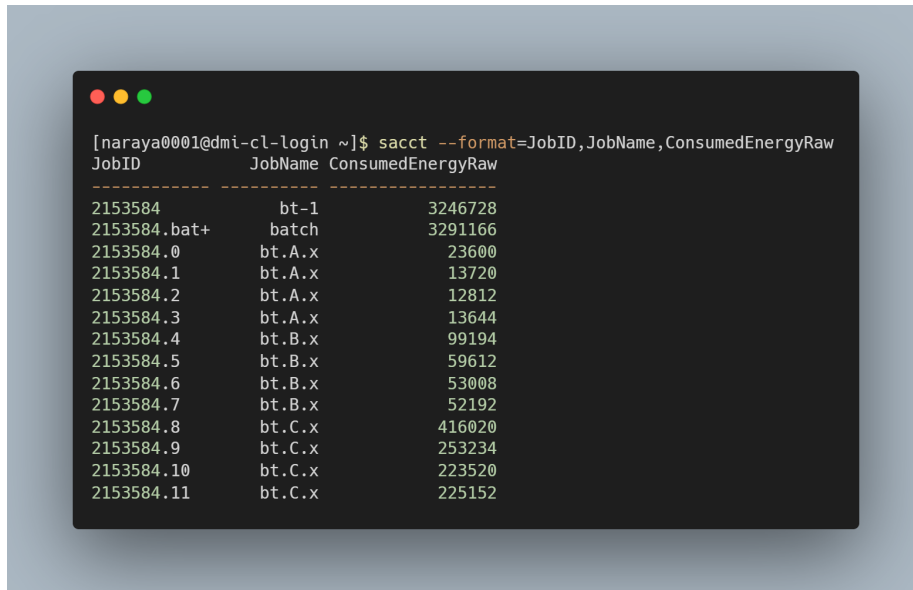
- The collected energy consumption data is integrated with SLURM’s accounting system.
- This integration allows associating energy consumption with specific jobs or users, enabling insights into resource usage and energy costs.
- Tools like ‘sacct’ and ‘sreport’ can be used to report on energy consumption alongside traditional job metrics like CPU time or memory usage. We have used **sacct** extensively in this thesis to retrieve the energy reported by SLURM as shown in the Figure 4.8

Benefits

- **Improved Resource Management:** By monitoring energy consumption, administrators can identify resource allocation strategies that optimize performance while minimizing energy usage.
- **Cost Awareness:** Energy usage reports can help users and administrators understand the energy footprint of their jobs and promote energy-efficient practices.
- **Power Capping:** SLURM offers features for setting power caps on individual nodes or the entire cluster, allowing for control over energy expenditure.

Limitations[43]

- **Hardware Dependence:** Functionality relies on compatible hardware with features like IPMI or RAPL for in-band data acquisition.



```
[naraya0001@dmi-cl-login ~]$ sacct --format=JobID,JobName,ConsumedEnergyRaw
JobID      JobName      ConsumedEnergyRaw
-----
2153584    bt-1         3246728
2153584.bat+  batch       3291166
2153584.0    bt.A.x       23600
2153584.1    bt.A.x       13720
2153584.2    bt.A.x       12812
2153584.3    bt.A.x       13644
2153584.4    bt.B.x       99194
2153584.5    bt.B.x       59612
2153584.6    bt.B.x       53008
2153584.7    bt.B.x       52192
2153584.8    bt.C.x       416020
2153584.9    bt.C.x       253234
2153584.10   bt.C.x       223520
2153584.11   bt.C.x       225152
```

Figure 4.8: `sacct` command to retrieve the energy consumed in Joules with JobID and JobName

- **Plugin Configuration:** Setting up plugins might require configuration specific to the hardware and monitoring infrastructure.
- **Data Accuracy:** The accuracy of energy data can vary depending on the chosen plugin and underlying hardware capabilities.

Overall, SLURM’s [43] [2] energy monitoring capabilities provide valuable insights into resource utilization and energy consumption in HPC environments. This information can be used to optimize job scheduling, promote energy-efficient practices, and ultimately reduce operational costs.

4.3.4 Prometheus

Prometheus[34] is a monitoring and alerting toolkit that is open-source, and it is designed for reliability and scalability. It is widely used in cloud-native environments and provides a flexible platform for monitoring various components of distributed systems.

Key features of Prometheus include[34]:

- **Multi-dimensional data model[34]:** Prometheus stores time-series data with key-value pairs, allowing flexible querying and aggregation based on various dimensions such as instance, job, and labels.
- **PromQL[34]:** Prometheus Query Language (PromQL) allows users to query and manipulate time-series data easily. It supports functions for aggregation, filtering, and mathematical operations.

- **Scalability**[34]: Prometheus is horizontally scalable, allowing it to handle large volumes of metrics from thousands of nodes. It uses a pull-based model, where Prometheus scrapes metrics from instrumented targets at regular intervals.
- **Service discovery**[34]: Prometheus supports various service discovery mechanisms, including static configuration, DNS-based discovery, and integrations with cloud providers such as Kubernetes.
- **Alerting**[34]: Prometheus provides built-in support for alerting based on defined alerting rules. It can send alerts via email, PagerDuty, or other alerting systems.
- **Integration**[34]: Prometheus can be integrated with other monitoring systems and data visualization tools. It has a rich ecosystem of exporters for collecting metrics from different services and applications.

Prometheus is commonly used for monitoring infrastructure, applications, and services in cloud-native environments. It helps operators and developers gain insights into system performance, diagnose issues, and ensure reliability and scalability of distributed systems. In the Figure 4.9, we can see the a basic Prometheus architecture.

Prometheus Components[33]

1. **Rules:** Rules in Prometheus are configurations that define conditions or expressions based on Prometheus Query Language (PromQL). These rules are evaluated periodically against the collected time-series data. If the condition defined in a rule is met, Prometheus generates an alert. Rules allow users to define complex monitoring scenarios and trigger alerts based on specific thresholds or conditions. Rules can be configured directly in the Prometheus server configuration file.
2. **Alertmanager:** Alertmanager is a separate component that manages alerts generated by Prometheus. It receives alerts from Prometheus server and then handles them based on configured rules and notification settings. Alertmanager supports various features such as deduplication, grouping, silencing, and routing of alerts to different receivers like email, PagerDuty, Slack, or custom webhooks. It enhances the flexibility and manageability of alerting in Prometheus.
3. **Scrape Discovery:** Scrape discovery is a mechanism in Prometheus for dynamically discovering and monitoring targets without manual configuration. Prometheus supports several service discovery mechanisms, including Kubernetes service discovery, DNS-based service discovery, EC2 auto-discovery, and more. These mechanisms allow Prometheus to automatically discover and monitor new instances of services, containers, or applications as they are added or removed from the environment. Scrape discovery simplifies the monitoring setup in dynamic and ephemeral environments like container orchestration platforms.
4. **Services:** In the context of Prometheus, services refer to the targets or endpoints that Prometheus monitors and collects metrics from. These services can be applications, containers, databases, servers, or any other system or component that exposes metrics in a format Prometheus understands. Prometheus uses various service discovery mechanisms to dynamically discover and monitor these services. Exporters are often used to expose metrics from services that do not natively support Prometheus metrics format.

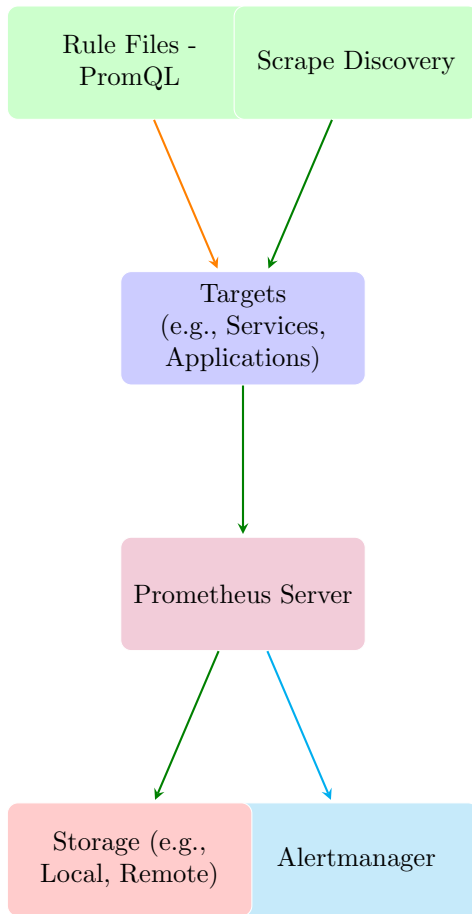


Figure 4.9: An illustration of the Prometheus architecture

5. **Storage:** Storage in Prometheus refers to the persistence layer where collected time-series data is stored. Prometheus uses a local on-disk storage engine by default, storing data in a custom time-series database format optimized for fast querying and retrieval. The storage engine efficiently compresses and indexes the time-series data to optimize disk usage and query performance. Prometheus also supports remote storage integrations with systems like Thanos or long-term storage solutions like Amazon S3 or Google Cloud Storage for storing historical data beyond the retention period of the local storage.

Prometheus is deployed on sciCORE for monitoring the nodes in the cluster and it is presented as <https://metrics.unibas.ch> which is only available at the sciCORE building and is not public. The Figure 4.10 represents the power consumption when the application bt-B was ran for 10 iterations. This graph is represented when the query is used as shown in the Listing 4.1. We can make a command-line request using curl to query a Prometheus server for monitoring data as shown in the Listing 4.2.

```

1 {
2   "status": "success",
3   "data": {
4     "resultType": "vector",
5     "result": [
6       {
7         "metric": {
8           "__name__": "telegraf_ipmi_sensor",
9           "_name": "sys_power",
10          "host": "sgi51.cluster.bc2.ch",
11          "instance": "localhost:9108",
12          "job": "graphite",
13          "unit": "watts"
14        },
15        "value": [
16          1709199580.732,
17          "450"
18        ]
19      }
20    ]
21  }
22 }

```

Listing 4.3: Json Response from Prometheus server

```

1 telegraf_ipmi_sensor{host="sgi51.cluster.bc2.ch",unit="watts"}

```

Listing 4.1: Query to extract power consumption in prometheus for node sgi51

```

1 curl -sG "https://metrics.scicore.unibas.ch/prometheus/api/v1/query" --data
   -urlencode "query=telegraf_ipmi_sensor{host=\"sgi51.cluster.bc2.ch\",
   unit=\"watts\"}"

```

Listing 4.2: Curl command to query Prometheus

4.3.5 Ganglia

“Ganglia” [31] is a monitoring system designed to be scalable and distributed. It is mainly used to track high-performance computing systems, clusters, and grids. It provides real-time monitoring, trending, and visualization of system and application performance metrics. The architecture of Ganglia [7] is designed to be modular, flexible, and highly scalable. Below is a description of its essential components and their roles:

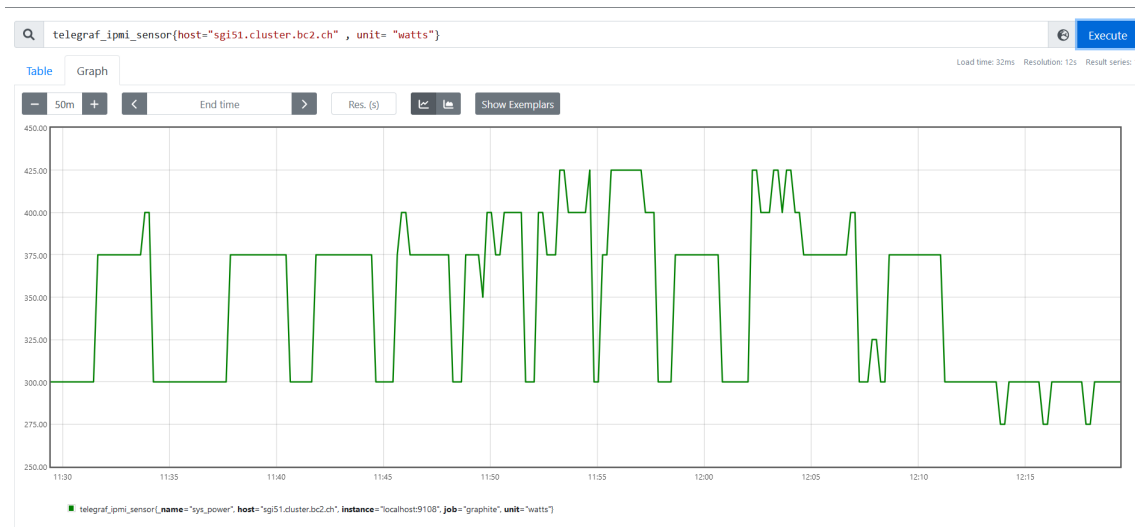


Figure 4.10: Power consumption of bt-B on node sgi51 in Prometheus at sciCORE

1. Ganglia Monitoring Daemon (gmond) [30]:

- The gmond daemon runs on each node (or host) in the monitored cluster.
- It collects local system metrics, such as CPU utilization, memory usage, network activity, disk I/O, and custom application metrics.
- gmond aggregates the collected metrics and periodically sends them to one or more Ganglia metadata collection points (gmetad).

2. Ganglia Metadata Daemon (gmetad)[30]:

- The **gmetad** daemon collects and stores metrics received from multiple gmond daemons in a centralized location.
- It organizes the metrics into a hierarchical tree-like structure based on the cluster's logical and physical layout.
- **gmetad** can store the aggregated metrics in a Round-Robin Database (RRD) format for historical trend analysis.
- It provides a web-based interface for querying and visualizing the collected metrics, known as the Ganglia Web Frontend.

3. Ganglia Web Frontend[30]:

- It is a web-based graphical user interface for accessing monitoring data.
- Users can view real-time and historical performance metrics for hosts, clusters, or grid resources.
- Users can customize dashboards, create graphs, set alerts, and analyze trends using the interactive interface.

- The front end retrieves monitoring data from **gmetad** and presents it in a user-friendly format using HTML, JavaScript, and CSS.

4. Hierarchical Structure[30]:

- Ganglia organizes monitored entities (hosts, clusters, grids) into a hierarchical tree structure.
- Each node in the hierarchy represents a cluster or a group of hosts, with child nodes representing individual hosts or sub-clusters.
- This hierarchical structure allows users to navigate and manage monitoring data effectively, particularly in large-scale distributed environments.

5. Scalability and Fault Tolerance[30]:

- Ganglia’s architecture is designed for scalability and fault tolerance.
- It can monitor thousands of hosts and metrics in distributed computing environments.
- Multiple gmond daemons can send metrics to multiple gmetad instances, providing redundancy and load balancing.
- Ganglia’s decentralized architecture minimizes single points of failure and ensures robustness in large-scale deployments.

Overall, Ganglia’s architecture[30] provides a comprehensive solution for monitoring and managing the performance of complex distributed computing environments. It offers real-time insights into system and application behavior for administrators and users alike. In the Figure 4.12 we can see the power consumption of application bt-A with increasing number of threads in OpenMP.

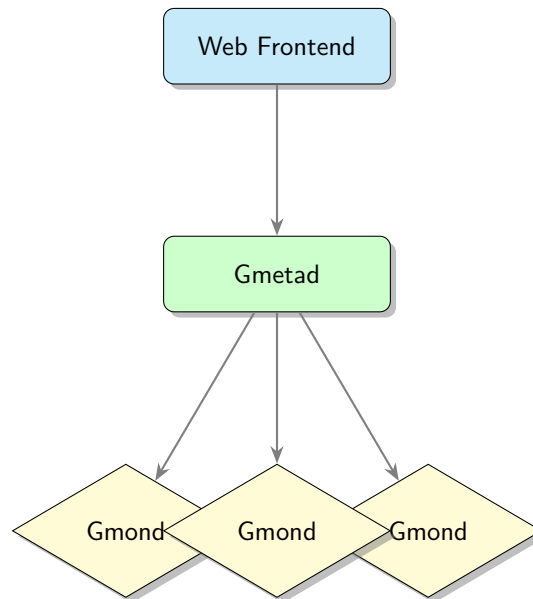


Figure 4.11: Ganglia Architecture Diagram

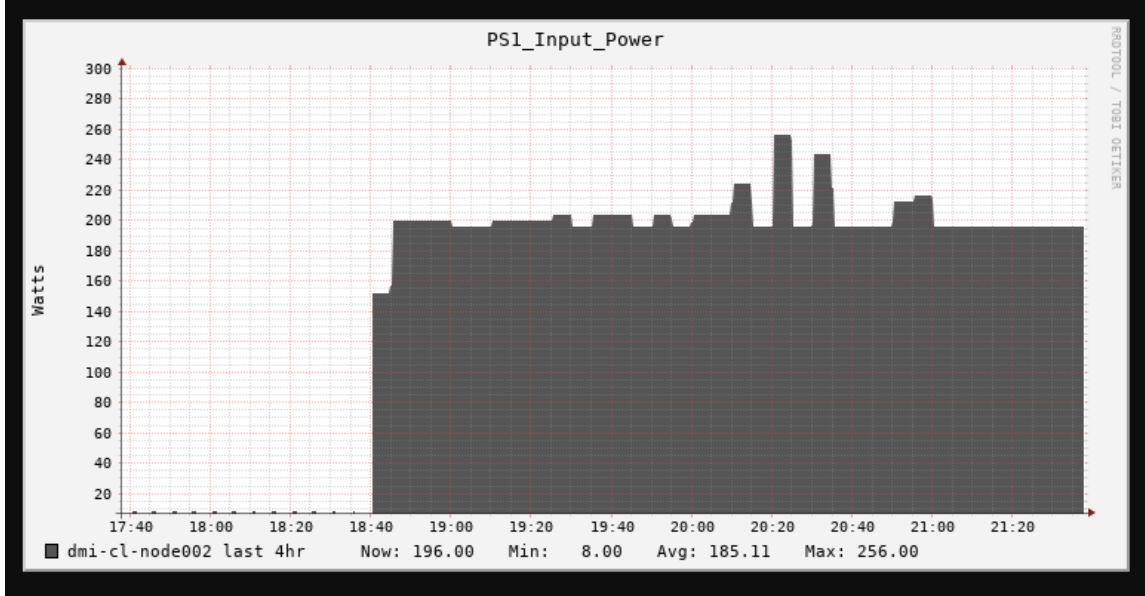


Figure 4.12: Power consumption of bt-A with increasing number of threads over Ganglia

4.4 Green Algorithms Code for System Wide Carbon Footprint Generation

Green Algorithms Code [27] has been used in this thesis to generate the carbon footprint of the user on the High Performance Computing cluster. The code provides a `cluster.yaml` file that has to be updated with the hardware information of the nodes on the cluster as shown in the Figure 4.13 where you need to update the CPU and GPU model information for the specific partitions and their TDP (Thermal Design Power). The script works for SLURM as a workload manager and script could be modified for other workload managers.

The carbon footprint of an algorithm[27] is determined by estimating the energy consumption of the algorithm and multiplying it by the carbon intensity associated with producing that energy in a particular location:

To calculate the carbon footprint, the following formula is utilized: To calculate the carbon footprint, the following formula is utilized:

$$\text{Carbon footprint} = \text{Energy consumption} \times \text{Carbon intensity} \quad (4.1)$$

The energy consumption is calculated as follows:

$$\text{Energy consumption} = \text{Runtime} \times (\text{Power draw for cores} \times \text{Usage} + \text{Power draw for memory}) \times \text{PUE} \times \text{PSF} \quad (4.2)$$

Here, the power draw for the computing cores depends on the CPU model and the number of cores, while the power draw for memory depends solely on the memory size. The usage factor

```
cluster_name: "hpc-cluster" # [str]
granularity_memory_request: 6 # [number] in GB,
partitions: # a list of the different partitions on the cluster
  xeon: # name of the partition
    type: CPU # [CPU or GPU]
    model: "Intel(R) Xeon(R) E5-2640 @ 2.40 GHz" # [str] the model of the processing core on this
    partition
    TDP: 90 # [number] TDP of the processor, in W, per core

  amd: # name of the partition
    type: GPU # [CPU or GPU]
    model: "AMD EPYC 7742 64-Core Processor" # [str] the model of the processing core on this partition
    TDP: 280 # [number] TDP of the processor, in W, per core

  gpu: # name of the partition
    type: CPU # [CPU or GPU]
    model: "Intel® Xeon® Gold 6258R Processor @ 2.70 GHz" # [str] the model of the processing core on
    this partition
    TDP: 205 # [number] TDP of the processor, in W, per core

PUE: 1.67 # [number > 1] Power Usage Effectiveness of the facility
CI: 467 # [number] carbon intensity of the geographic location, in gCO2e/kWh
```

Figure 4.13: Sample information to be updated on cluster YAML

(defaulted to 1 for full usage) adjusts for the actual core usage. PUE (Power Usage Effectiveness) represents the additional energy required to operate the data center (e.g., cooling, lighting). PSF (Pragmatic Scaling Factor) is applied for multiple identical runs, such as testing or optimization.

The carbon intensity is influenced by the location and the methods used to generate electricity. It is important to note that the "energy consumption" mentioned above is location-independent. One of the sample results for the same is shown in the the Figure 4.14

```
#####  
#  
# Your carbon footprint on hpc-cluster #  
# (2023-09-01 / 2024-01-24) #  
#  
#####  
  
-----  
| 14 kgCO2e |  
-----  
  
...This is equivalent to:  
- 14 tree-months  
- driving 78 km  
- 0.27 flights between Paris and London  
  
...24.4% of your jobs failed, which represents a waste of 1 kgCO2e (1.32 tree-months).  
..On average, you request at least 9.9 times the memory you need. By only requesting the memory  
you needed, you could have saved 779 gCO2e (0.85 tree-months).  
  
Energy used: 29.08 kWh  
- CPUs: 16.31 kWh (56%)  
- GPUs: 0.00 kWh (0%)  
- Memory: 1.10 kWh (4%)  
- Data centre overheads: 11.67 kWh (40%)  
Carbon intensity used for the calculations: 467 gCO2e/kWh  
  
Summary of your usage:  
- First/last job recorded on that period: 2024-01-03/2024-03-23  
- Number of jobs: 897 (678 completed)  
- Core hours used/charged: 644.3 (CPU), 0.0 (GPU), 644.3 (total).  
- Total usage time (i.e. when cores were performing computations):  
  - CPU: 7 days, 13:14:52.269000  
  - GPU: 0:00:00  
- Total wallclock time: 1 day, 19:46:50  
- Total memory requested: 63,364 GB
```

Figure 4.14: Sample carbon footprint generated by Green Algorithms code

4.5 Experimental Setup

Factor	Values	Properties
Applications	BT SP LU	Block Tri-diagonal Solver Scalar Penta-Diagonal Solver Lower Upper Gauss-Siedel Solver
Problem Size	A, B, C	Standard test problems; 4X size increase going from one class to the next
Paradigms	OpenMP (T) 1, 2, 4, 8, 16, 20 MPI (P): 1, 4, 9, 16, 25 and 36 Hybrid (P:T) 1:16, 2:8, 8:2, 16:1	Increasing the number of threads Increasing number of processes with 1 node and 2 nodes The ratio of processes to threads
System	miniHPC sciCORE	partition: xeon partitions: rtx-8000 and a100
Energy Consumption Metrics	Node Level in Watts	miniHPC: Ganglia sciCORE: Prometheus
	Job Level in Joules	miniHPC: SLURM sciCORE: SLURM is disabled
	Application Level Joules	miniHPC: PMT and Likwid sciCORE: PMT and Likwid
	System Wide in kWh	miniHPC: Green Algorithm sciCORE: Green Algorithms
CPU Frequency	low, medium, high, highm1	Different CPU frequency levels
Repetition	10	Repeated Experiments for mean and median

Table 4.2: Experimental setup for factorial experiments.

Applications

- Three applications are considered: BT (Block Tri-diagonal Solver), SP (Scalar Penta-Diagonal Solver), and LU (Lower Upper Gauss-Siedel Solver). These represent different computational workloads with varying characteristics.

Problem Size:

- Three standard test problems denoted as A, B, and C are used.
- Each problem size is increased by a factor of 4X going from one class to the next, providing a range of problem complexities to assess performance and energy consumption.

Paradigms:

- **OpenMP (T):** Shared-memory parallelism is explored with varying numbers of threads (1, 2, 4, 8, 16, 20).
- **MPI (P):** Distributed-memory parallelism is examined with varying numbers of processes (1, 4, 9, 16, 25, 36) on one node and two nodes.
- **Hybrid (P:T):** Combination of OpenMP and MPI paradigms with different ratios of processes to threads (1:16, 2:8, 8:2, 16:1), providing insights into mixed parallelism strategies.

System:

- Two HPC Systems are utilized:
 - miniHPC: Utilizes Xeon processors.
 - sciCORE: Utilizes RTX-8000 and A100 processors.
- The presence of different hardware configurations may influence performance and energy consumption.

Energy Consumption Metrics:

- Energy consumption is measured at multiple levels:
 - Node Level: Wattage consumption at the node level with Ganglia and Prometheus.
 - Job Level: Total energy consumed per job in Joules via SLURM.
 - Application Level: Energy consumption specific to each application in Joules with PMT and LIKWID.
 - System Wide: Total energy consumption in kWh, providing an overview of system-wide energy usage.

CPU Frequency:

- Various CPU frequency levels are investigated, including low, medium, high, and highm1, to understand how CPU performance states affect energy consumption and performance.

Repetition:

- Each experiment is repeated 10 times to ensure statistical significance and calculate mean and median values for performance and energy consumption metrics.

Total Experiments:

- The total number of experiments is calculated by multiplying the number of values for each factor with the number of repetitions, resulting in 2700 experiments.

This setup is designed to comprehensively evaluate the performance and energy consumption characteristics of different parallel computing paradigms under varying conditions, providing insights into their suitability for different applications and problem sizes on diverse HPC systems.

Chapter 5

Results

In this section, the results of the study on the energy consumption of NAS benchmarks using three programming paradigms: OpenMP, MPI, and Hybrid (a combination of OpenMP and MPI) are presented. Data has been collected at the node level, job level, and application level. Given the extensive nature of the study, which involved close to 2700 experiments, only relevant data will be included in the results section. Repetitive data will be available in the appendix of this thesis.

5.1 OpenMP Experiments

5.1.1 miniHPC and sciCORE

For the OpenMP programming paradigm on the miniHPC system, experiments were conducted by varying the number of threads from 1 to 20. The energy consumption trends for each application and program size combination are depicted in Figure 5.1 for the miniHPC cluster and the Figure 5.2 for sciCORE Cluster. The graphs in both the figures illustrates the comparison of mean energy consumption versus the number of threads for the NAS benchmark **BT (Block Tri-diagonal Solver)** across three different job scheduling systems: PMT, SLURM, and Ganglia and Prometheus for sciCORE. Each line represents the mean/median energy consumption as the number of threads varies from 1 to 20.

- The blue line represents the mean/median energy consumption for PMT, showing a noticeable decrease as the number of threads increases.
- The orange line depicts the mean/median energy consumption for SLURM, which follows a similar decreasing trend with increasing thread count, albeit at a different rate compared to PMT. This is disabled on sciCORE so we do not have the SLURM energy data for sciCORE
- The green line represents the mean/median energy consumption for Ganglia in miniHPC and Prometheus in sciCORE. It shows a decrease in energy consumption with increasing thread count, although with slightly different fluctuations compared to PMT and SLURM.

Additionally, annotations on the Ganglia line indicate the corresponding mean time values, providing insights into the relationship between computational efficiency and energy consumption. It

is observed that as the mean/median time decreases, there is a corresponding decrease in mean/-median energy consumption, indicating improved computational efficiency.

5.2 MPI Experiments

5.2.1 miniHPC

The graphs in the Figure 5.3 compares the mean energy consumption versus the number of processes for the NAS benchmark **LU (Lower-Upper Gauss-Seidel solver)** across three different job scheduling systems: PMT, SLURM, and Ganglia.

- The blue line represents the mean energy consumption for PMT, which exhibits a decreasing trend as the number of processes increases.
- The orange line depicts the mean energy consumption for SLURM, showing a similar decreasing trend with increasing process count, albeit at a different rate compared to PMT.
- The green line represents the mean energy consumption for Ganglia, showcasing a decrease in energy consumption with increasing process count, although with slightly different fluctuations compared to PMT and SLURM.

Annotations on the Ganglia line indicate the corresponding mean time values, providing insights into the relationship between computational efficiency and energy consumption. It is observed that as the mean time decreases, there is a corresponding decrease in mean energy consumption, indicating improved computational efficiency.

5.2.2 sciCORE

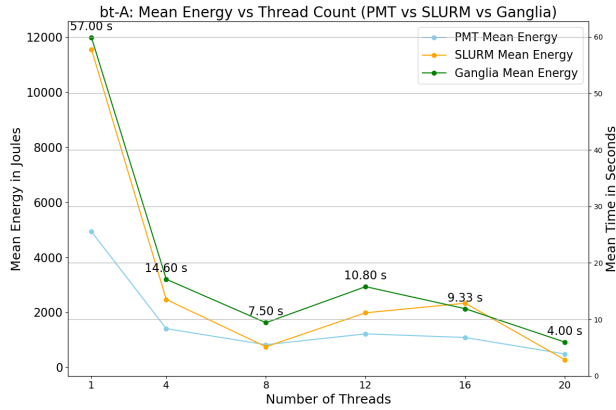
Similarly, experiments on the sciCORE system revealed exactly the same energy consumption trends for OpenMP with varying thread counts as show in the Figure 5.6.

- The blue line represents the mean energy consumption for PMT, which exhibits a decreasing trend as the number of processes increases.
- Since SLURM is disabled on sciCORE we do not have SLURM energy data here.
- The green line represents the mean energy consumption for Prometheus, showcasing a decrease in energy consumption with increasing process count.

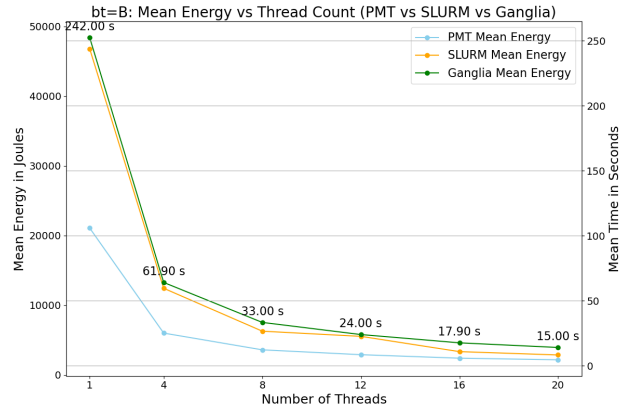
5.2.3 MPI experiments with 2 nodes

The graph the Figure 5.7 illustrates the relationship between the number of processes (1, 4, 9, 16, 25 and 36) and energy consumption, as well as runtime, for three different scenarios (SP-A, SP-B, and SP-C) in an MPI (Message Passing Interface) setting across two nodes cl-node002 and cl-node003.

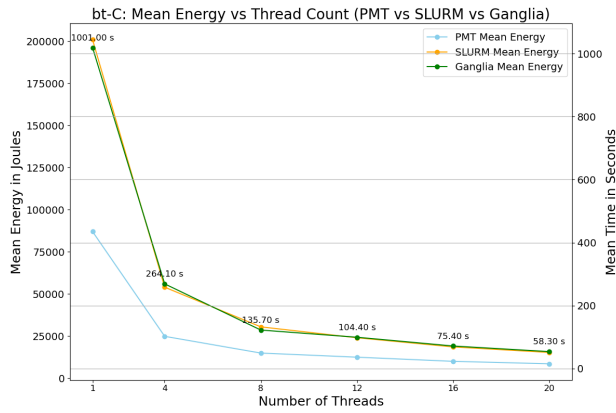
For each scenario:



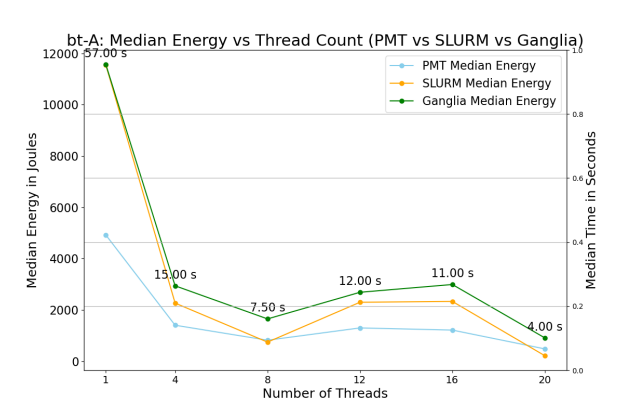
(a) bt-A: Thread Count vs Mean Energy



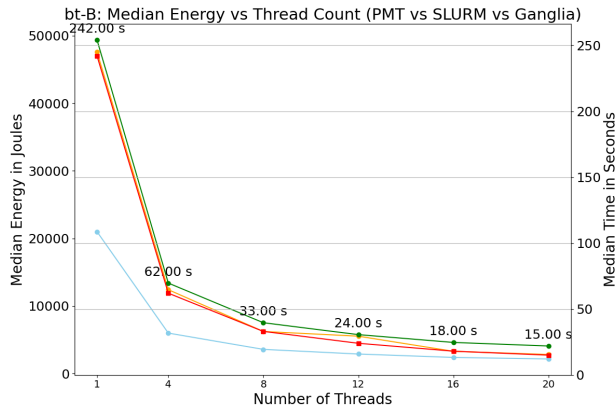
(b) bt-B: Thread Count vs Mean Energy



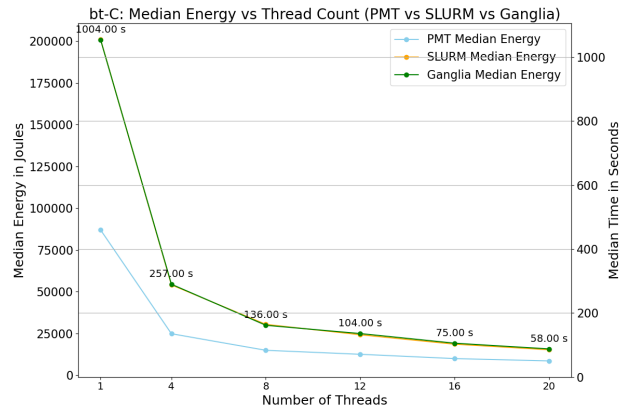
(c) bt-C: Thread Count vs Mean Energy



(d) bt-A: Thread Count vs Median Energy

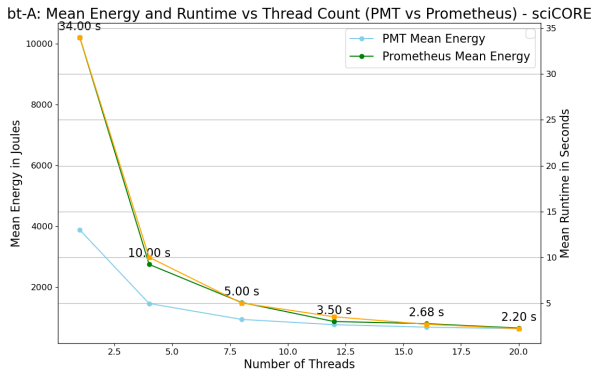


(e) bt-B: Thread Count vs Median Energy

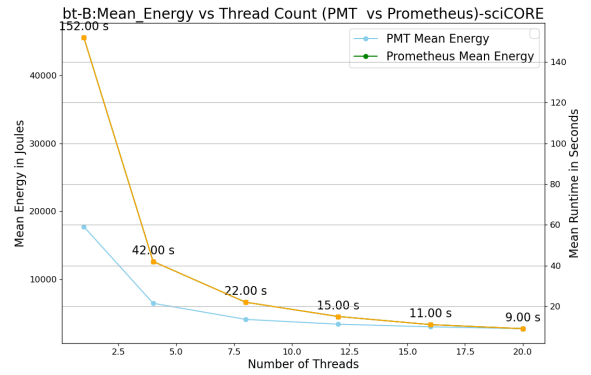


(f) bt-C: Thread Count vs Median Energy

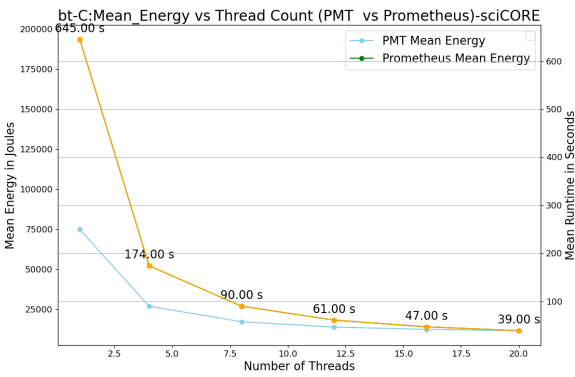
Figure 5.1: bt Application on MiniHPC for OpenMP



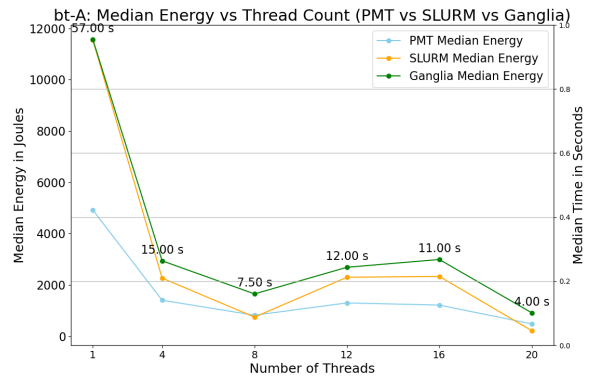
(a) bt-A: Thread Count vs Mean Energy



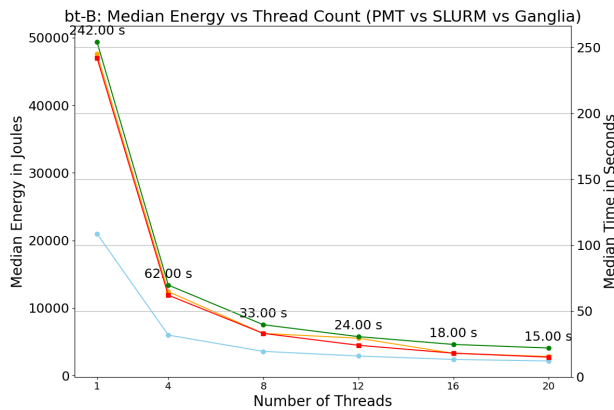
(b) bt-B: Thread Count vs Mean Energy



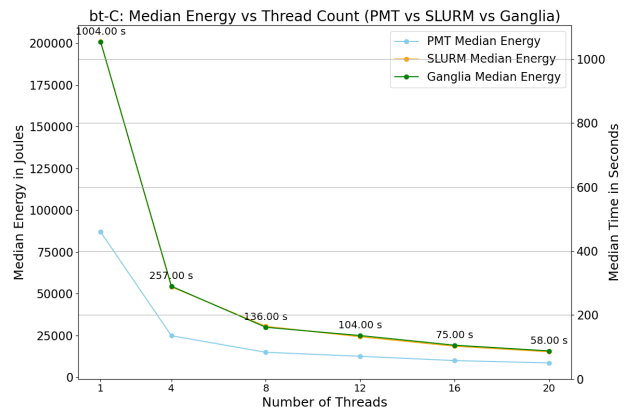
(c) bt-C: Thread Count vs Mean Energy



(d) bt-A: Thread Count vs Median Energy

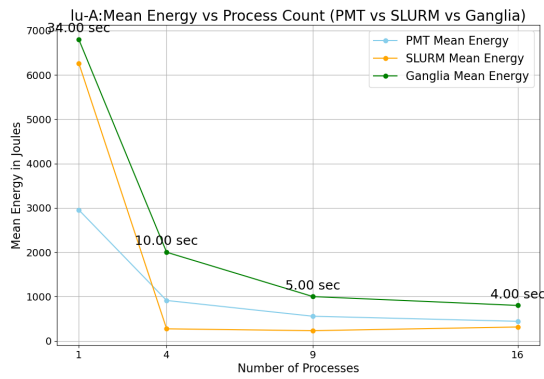


(e) bt-B: Thread Count vs Median Energy

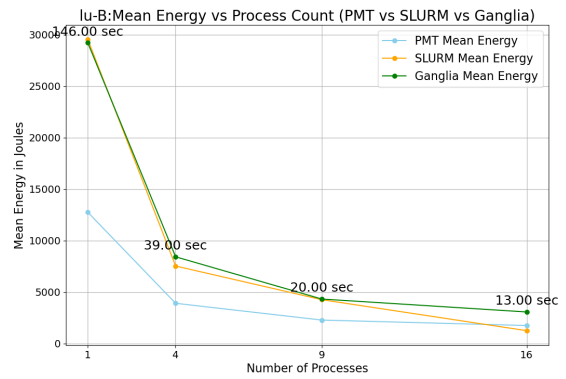


(f) bt-B: Thread Count vs Median Energy

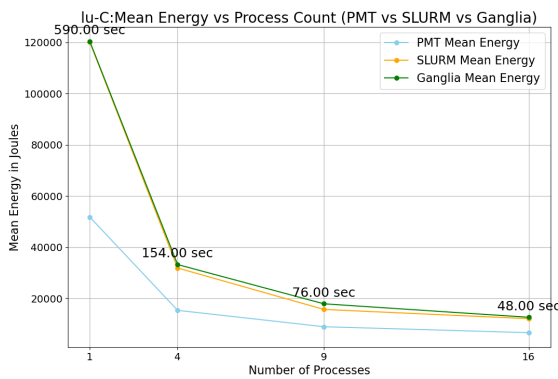
Figure 5.2: bt Application on sciCORE for OpenMP



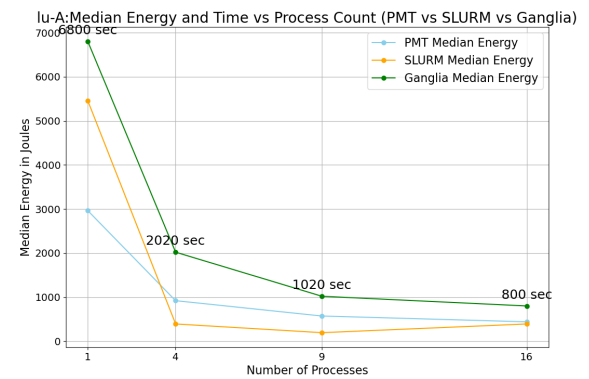
(a) LU-A: Process Count vs Mean Energy



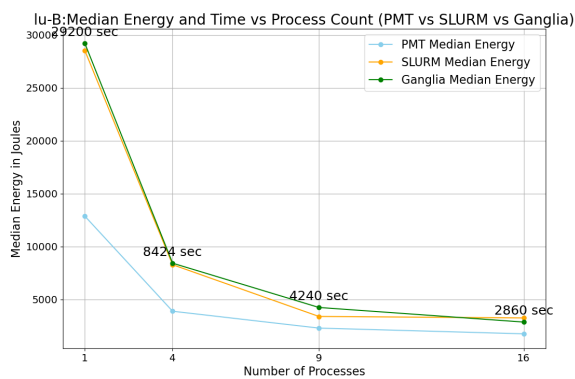
(b) LU-B: Process Count vs Mean Energy



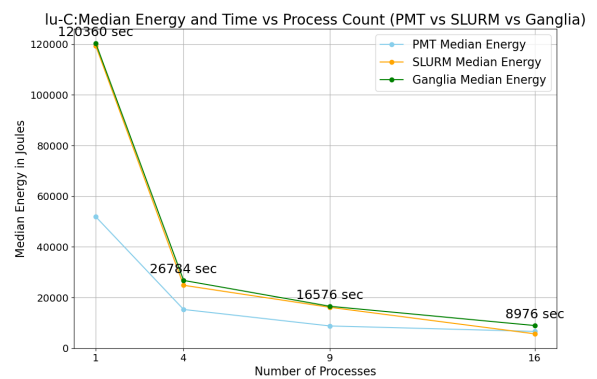
(c) LU-C: Process Count vs Mean Energy



(d) LU-A: Process Count vs Median Energy

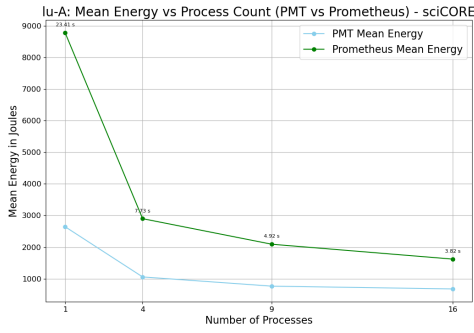


(e) LU-B: Process Count vs Median Energy

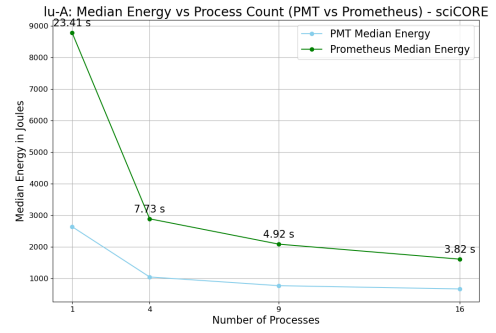


(f) LU-C: Process Count vs Median Energy

Figure 5.3: LU Application on MiniHPC for MPI for single node

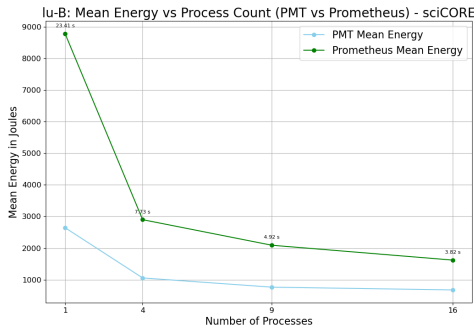


(a) Mean Energy for MPI in sciCORE

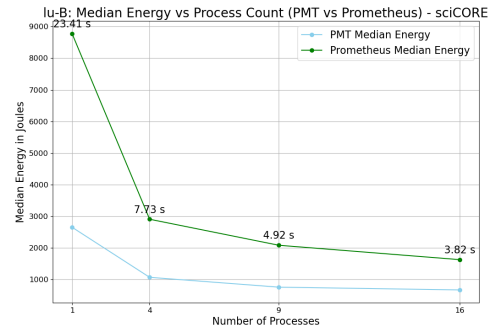


(b) Median Energy for MPI in sciCORE

Figure 5.4: LU for Class A, for single node, MPI

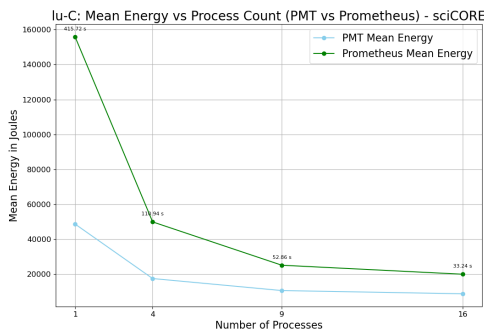


(a) Mean Energy for MPI in sciCORE

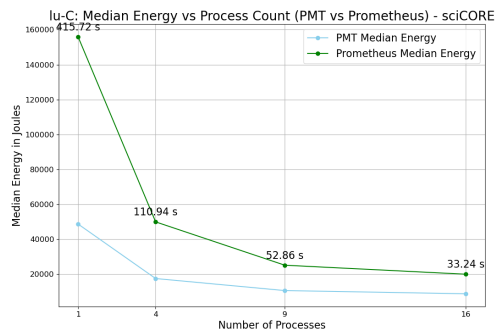


(b) Median Energy for MPI in sciCORE

Figure 5.5: LU for Class B, for single node, MPI

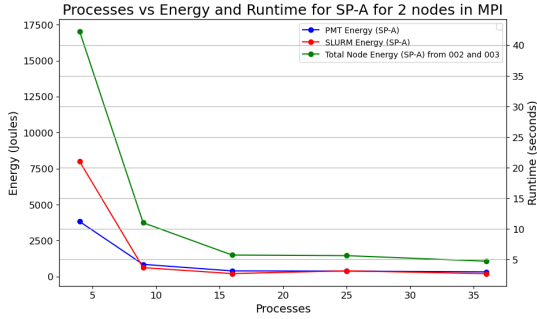


(a) Mean Energy for MPI in sciCORE

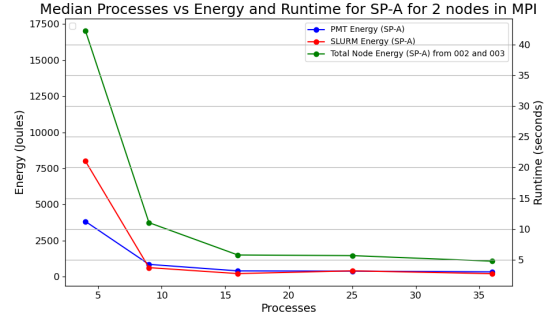


(b) Median Energy for MPI in sciCORE

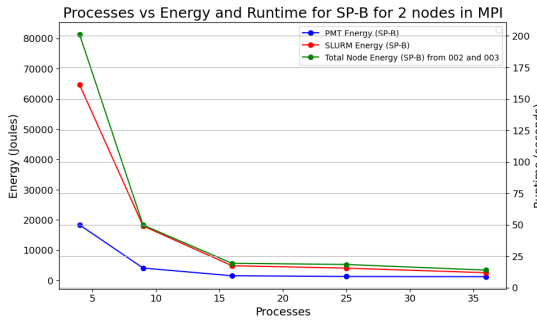
Figure 5.6: LU for Class C, for single node, MPI



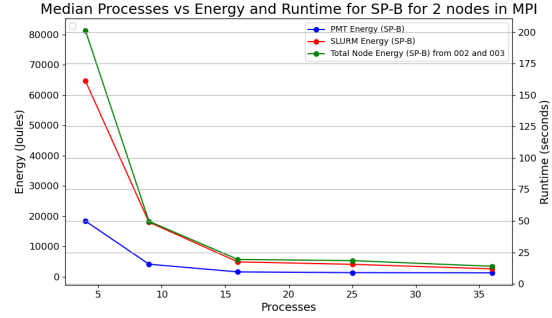
(a) SP-A : Mean Process vs Energy



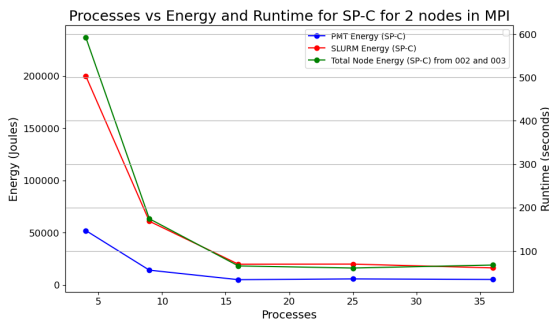
(b) SP-A : Median Process vs Energy



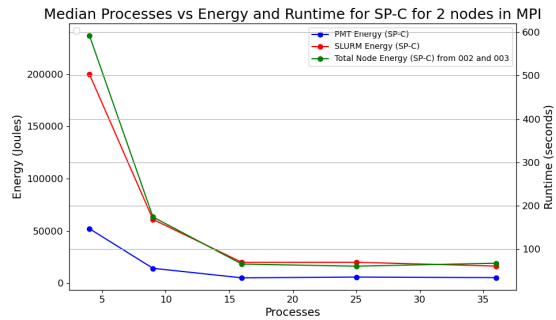
(c) SP-B : Mean Process vs Energy



(d) SP-B : Median Process vs Energy



(e) SP-C : Mean Process vs Energy



(f) SP-C : Median Process vs Energy

Figure 5.7: SP Application on miniHPC for MPI for 2 nodes

- The blue line represents the energy consumption measured by PMT (Performance Monitoring Tool).
- The red line represents the energy consumption measured by SLURM, a job scheduling system.
- The green line represents the total node energy consumption, calculated by summing the energy consumption of individual nodes (node_002 and node_003).

Additionally, each scenario includes runtime values plotted with a secondary y-axis (black line).
Observations:

- In SP-A, as the number of processes increases, there is a significant decrease in energy consumption, with a notable difference between PMT and SLURM measurements. Total node energy shows a similar trend, decreasing with increasing processes.
- In SP-B, a similar trend is observed, with a significant decrease in energy consumption as processes increase, again with differences between PMT and SLURM measurements. Total node energy follows a comparable pattern.
- In SP-C, energy consumption tends to decrease as the number of processes increases, with variations between PMT and SLURM measurements. Total node energy reflects this trend.

Overall, the graphs provide insights into the energy consumption patterns across different process configurations in MPI settings, aiding in understanding resource utilization and optimization strategies for parallel computing applications.

5.3 Hybrid Experiments

The graphs in the Figure 5.8 illustrate the performance metrics (time and energy consumption) for three different LU benchmarks (LU-A, LU-B, and LU-C) across various process-to-thread configurations. Each subplot represents one benchmark, with one subplot for time and another for energy consumption.

In each subplot:

- The x-axis represents different process-to-thread configurations, where the format "x:y" denotes x processes and y threads.
- The y-axis represents the corresponding performance metric, either time (in seconds) or energy consumption (in Joules).

For each benchmark:

- Different process-to-thread configurations are represented by colored bars, with each color corresponding to a specific configuration.
- The height of each bar indicates the value of the performance metric (time or energy consumption) for the corresponding configuration.

Observations:

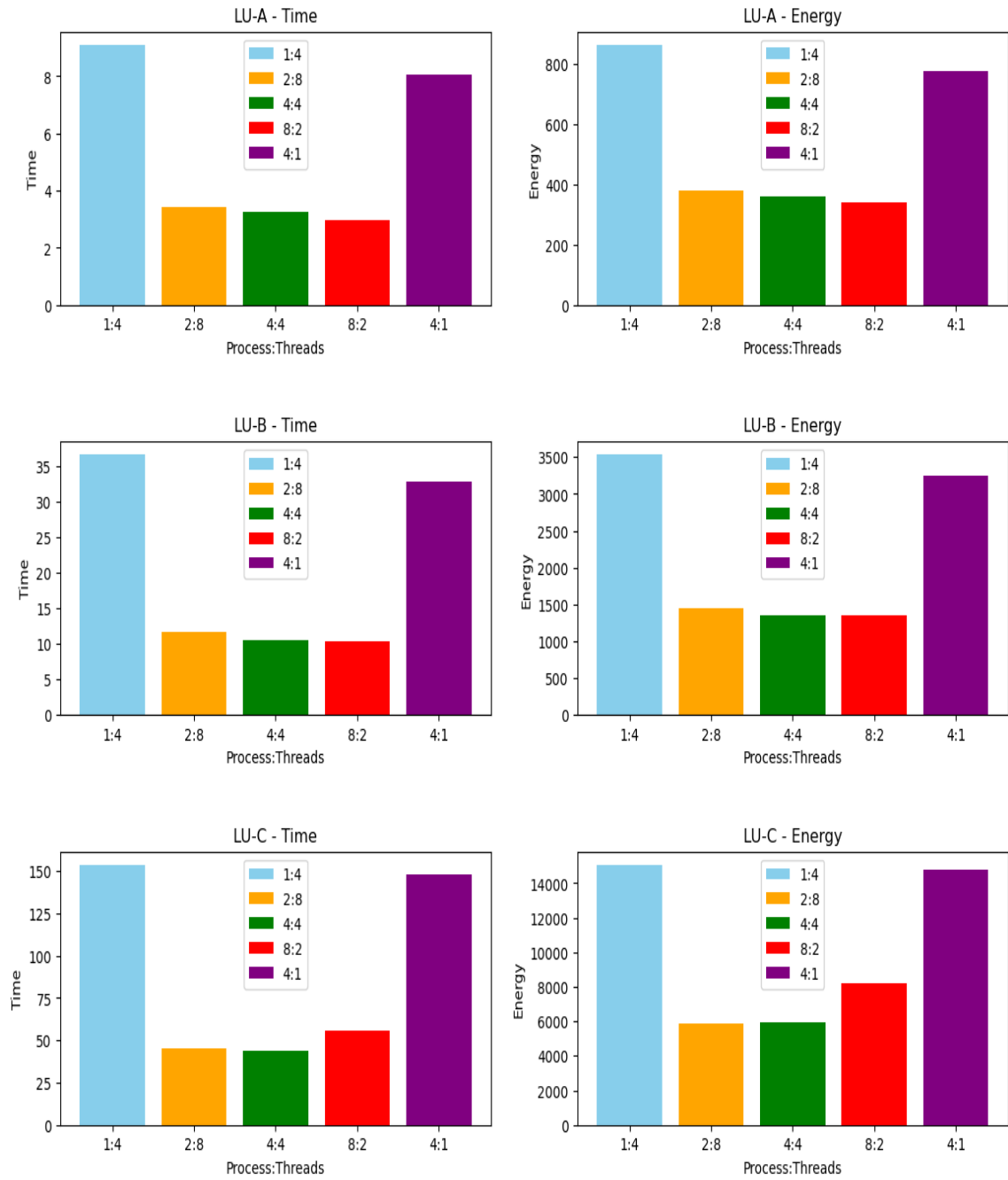


Figure 5.8: Hybrid (OpenMP+MPI) experiments for LU application

- For LU-A and LU-B benchmarks, configurations with higher thread counts generally exhibit lower time and energy consumption compared to configurations with fewer threads. This trend suggests improved parallel performance with increased thread utilization.
- In contrast, for the LU-C benchmark, the relationship between thread count and performance metrics is less consistent. While some configurations with higher thread counts show lower performance metrics, this trend is not as pronounced as in LU-A and LU-B. This variation may be due to differences in the computational workload and resource utilization characteristics of the LU-C benchmark.

Overall, the graph provides insights into the performance characteristics of LU benchmarks under different process-to-thread configurations, aiding in the optimization of resource allocation and workload distribution strategies for parallel computing tasks.

5.4 CPU Frequency vs Energy

This section describes the affect of changing frequency on power consumption at the frequencies, low, medium, highm1 and high. The graph in the Figure 5.9 illustrates the relationship between different frequencies and energy usage for various thread configurations (1, 4, 8, 12, 16, and 20 threads) in the **bt-C** benchmark. The trends remain the same for SP and LU application and the graphs are provided in the appendix.

Key Features:

- **Bar Plot:** Each group of bars represents a specific thread configuration, with different colors indicating different frequencies.
- **X-axis:** Indicates the frequencies (low, medium, highm1, high).
- **Y-axis:** Represents the energy usage (in Joules).
- **Legend:** Provides a key to interpret the colors, indicating the number of threads.
- **Annotations:** Each bar is annotated with the corresponding runtime (in seconds) to provide insights into the trade-off between energy usage and runtime.

The graph facilitates the analysis of energy consumption patterns across various frequencies and thread configurations, aiding in optimizing performance and resource utilization strategies for the bt-C benchmark.

5.5 Integration of all Energy components: The Utility Tool

The primary objective of this thesis revolves around the comprehensive integration of energy consumption data across different levels: application, job, and node. Leveraging the robust capabilities of LIKWID, a powerful performance monitoring tool, we aim to delve into the intricacies of energy usage at the application level.

The submitted SLURM job script defines specific parameters for job execution and initiates the execution of the **srun_energy** script on the allocated compute nodes. Within the **srun_energy** script, the Likwid module is loaded to facilitate the measurement of energy consumption and

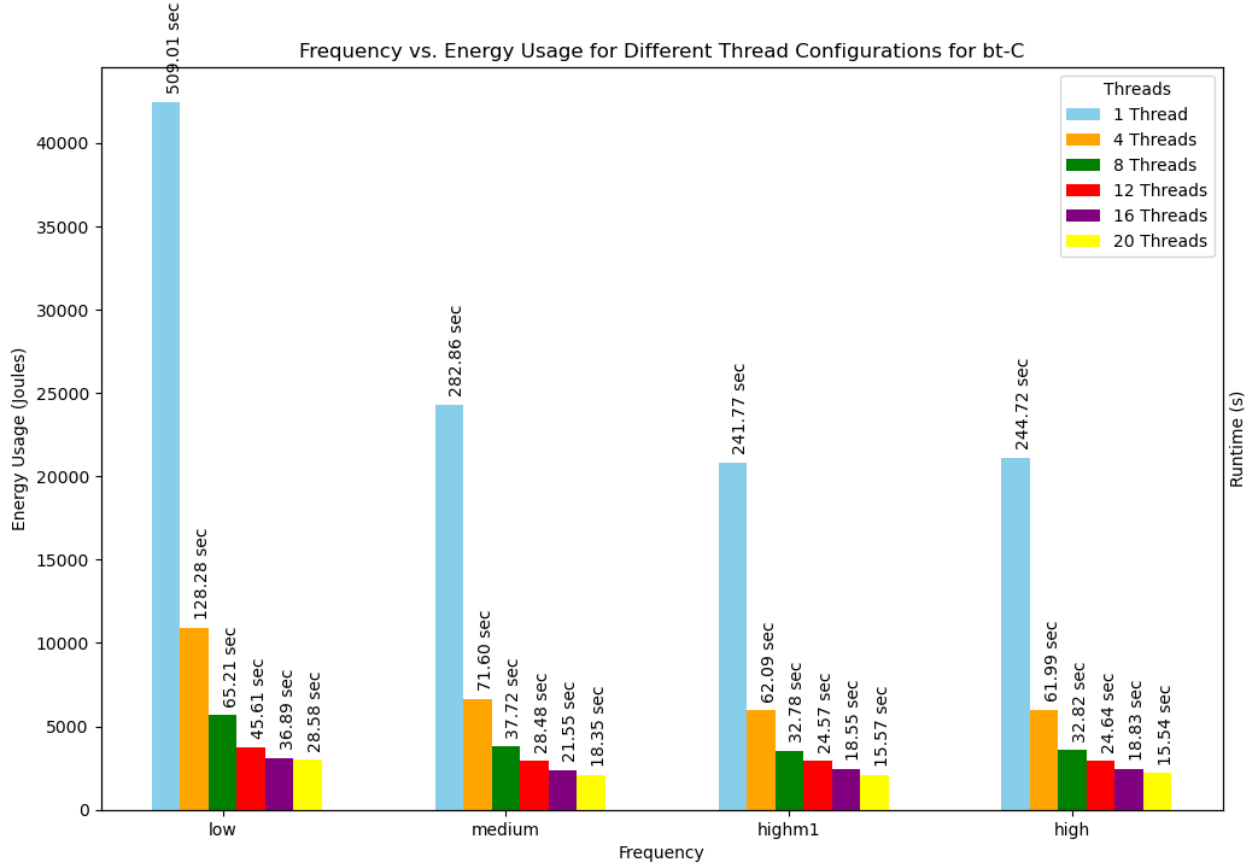


Figure 5.9: Frequency vs. PMT energy Usage for different thread configurations for bt-C

various performance metrics using Likwid’s versatile tools such as **likwid-powermeter** and **likwid-perfctr**. Additionally, the script captures real-time power consumption data from the SLURM environment.

The resulting data is meticulously stored in individual output files corresponding to each job. Notably, the extraction of current Watts from the SLURM **scontrol** information provides insights into node-level energy consumption. It’s worth mentioning that at sciCORE, we used Prometheus for extracting and analyzing node-level energy consumption data, thus enriching our understanding of energy dynamics at different levels of computation. The provided Bash script offers a systematic approach to extract energy-related data from various sources, contributing to a comprehensive understanding of energy consumption across different computational levels: application, job, and node.

- **File Number:** The script extracts a unique identifier for each dataset from the filename, facilitating data organization and identification.
- **Application-level Energy:** It captures energy consumption at the application level, pro-



```
[narya001@dmi-cl-login FinalScript]$ bash reporting
File number: 2153824, app-level-Energy: 2667.896000, job-level-joules: 10075], Node-level-Joules: 12000.820288
File number: 2153825, app-level-Energy: 2825.443170, job-level-joules: 9855], Node-level-Joules: 13447.147648
File number: 2153826, app-level-Energy: 2117.396635, job-level-joules: 11215], Node-level-Joules: 12056.695252
File number: 2153827, app-level-Energy: 1945.715023, job-level-joules: 4566], Node-level-Joules: 9864.316964
File number: 2153828, app-level-Energy: 2588.648620, job-level-joules: 7522], Node-level-Joules: 9954.012963
```

Figure 5.10: Utility Tool Showing Energy information at all levels

viding insights into resource utilization patterns and efficiency.

- **Socket-level Energy:** The script aggregates energy consumption at the socket level, offering hardware-specific energy usage metrics.
- **Node-level Power Consumption:** Real-time power consumption data is obtained, aiding in the assessment of energy utilization at the node level.
- **Job-level Energy Accounting:** Consumed energy data from SLURM job accounting is retrieved, providing a broader context for energy usage within the job environment.

This Bash script serves as a valuable tool for researchers and practitioners in analyzing and optimizing energy efficiency in high-performance computing environments, enabling informed decision-making regarding resource allocation and application performance. The Figure 5.10 shows the extracted energy at different level for the jobs run for the user.

Chapter 6

Discussion

6.1 OpenMP

When increasing the number of threads from 1 to 4 to 8 to 12 to 16 to 20, and observing a decrease in energy consumption, several conclusions can be drawn:

1. Efficiency Improvement: The decrease in energy consumption suggests that utilizing more threads leads to more efficient utilization of computational resources. This implies that parallelizing the computation among multiple threads allows for better resource utilization and potentially reduces overall energy consumption.

2. Power Consumption: As the number of threads increases and the computation is completed faster, the instantaneous power consumption during computation may increase due to higher computational intensity. However, since the overall energy consumption is decreasing, it indicates that the reduction in runtime outweighs the increase in power consumption, resulting in a net decrease in energy usage.

3. Scalability: The observed decrease in energy consumption with an increasing number of threads indicates good scalability of the application with respect to energy efficiency. This suggests that the application can effectively leverage additional computational resources provided by multi-threading without significantly increasing energy consumption.

Overall, the trend of decreasing energy consumption with an increasing number of threads implies that parallelization can contribute to energy-efficient execution of computational tasks, potentially leading to more sustainable and cost-effective computing solutions. Additionally, understanding the relationship between power consumption, computational intensity, and energy efficiency is essential for optimizing both performance and energy consumption in parallel computing applications.

6.2 MPI

When increasing the number of processes from 1 to 4 to 9 to 16 in MPI, and observing a decrease in energy consumption, several conclusions can be drawn:

- **Efficiency Improvement:** The decrease in energy consumption suggests that distributing the workload across more processes leads to more efficient utilization of computational resources. This indicates that parallelizing the computation allows for better resource utilization and potentially reduces the overall energy consumption.
- **Communication Overhead:** With more processes, there may be less communication overhead compared to fewer processes, leading to reduced energy consumption. This could be due to more efficient data exchange and synchronization mechanisms implemented in MPI as the number of processes increases.
- **Scalability:** The observed decrease in energy consumption as the number of processes increases indicates good scalability of the application with respect to energy efficiency. This suggests that the application can effectively leverage additional computational resources without significantly increasing energy consumption.

Overall, the trend of decreasing energy consumption with an increasing number of processes in MPI implies that parallelization can contribute to energy-efficient execution of computational tasks, potentially leading to more sustainable and cost-effective computing solutions.

- **Power and Energy Relationship:** Since power is the rate of energy consumption, when energy consumption decreases but the computation is completed in a shorter time (due to parallelization), the power consumption per unit time, or power, may increase. This is because the same amount of energy is consumed over a shorter period, leading to a higher instantaneous power consumption during computation.
- **Computational Intensity:** As the number of processes increases, the computational intensity (the amount of computation performed per unit time) also increases. This higher computational intensity results in higher power consumption during the computation phase, even though the overall energy consumption decreases.
- **Performance Efficiency:** The increase in power consumption with higher computational intensity can be seen as a trade-off for improved performance and reduced runtime. While the instantaneous power consumption may be higher, the overall energy consumption is lower due to the shorter runtime, resulting in better performance efficiency.

6.3 Hybrid

The decreased energy consumption in the combinations where the ratio of processes to threads is 2:8, 4:4 and 8:2 can be attributed to several factors:

1. **Reduced Context Switching:** With more threads per process (8:2), the workload can be distributed among the threads, minimizing the need for the operating system to constantly switch context between different processes. This context switching process itself consumes energy.
2. **Improved Cache Utilization:** Threads within a process share the same memory space, including the CPU cache. With more threads (8:2), there is a higher chance of relevant data residing in the cache, reducing the need to fetch data from slower main memory, which is more energy-intensive.

3. **Efficient Thread Management:** The operating system can schedule threads within a process more efficiently than managing separate processes. This reduces idle time for processing resources, leading to lower overall energy usage.

However, it is important to consider the specific workload:

- If the tasks are highly independent and do not share data, a higher number of processes (2:8) might be beneficial. This allows for better parallelism and utilization of multiple cores.
- If the tasks are tightly coupled and share data frequently, a higher number of threads (8:2) within a few processes might be more energy-efficient due to the factors mentioned above.

In conclusion: The optimal ratio for energy efficiency depends on the specific workload. In general, with workloads that benefit from thread-level parallelism and data sharing, a higher ratio of threads to processes (like 8:2) can lead to decreased energy consumption due to reduced context switching, improved cache utilization, and efficient thread management.

6.4 Affect of different CPU frequencies

The graphs 5.9 provides valuable insights into the energy consumption behavior of the system under different thread configurations and frequencies over miniHPC. Following points could explain the trends:

1. **Energy Efficiency with Thread Configurations:** The graph illustrates how the energy consumption at the node level changes with varying thread configurations. Generally, as the number of threads increases, the energy consumption decreases. This aspect we noticed in the OpenMP section
2. **Effect of Frequency on Energy Consumption:** The graph also showcases the impact of different frequencies on energy consumption. Higher frequencies typically result in lower energy consumption, indicating that tasks can be completed more quickly at higher clock speeds, thus reducing the overall energy usage.
3. **Diminishing Returns with Thread Counts:** While increasing the number of threads generally leads to reduced energy consumption, the graph suggests that there may be diminishing returns beyond a certain thread count. In other words, the energy efficiency gains become less significant as more threads are added. This phenomenon could be due to factors such as resource contention or overhead associated with managing a larger number of threads.
4. **Trade-off Between Energy Efficiency and Runtime:** It is important to note that while optimizing for energy efficiency, there's often a trade-off with computational speed as runtimes are better with increase in number of threads.

6.5 Corner Cases: NUMA effect

In high-performance computing (HPC), NUMA [23] zones refer to Non-Uniform Memory Access zones. NUMA zones arise in systems with multiple processors or cores, where each processor or

core has its own memory bank. However, NUMA zones specifically refer to Non-Uniform Memory Access zones within a single node or NUMA system. In a NUMA system, memory is physically distributed across multiple memory banks, and each processor or core has local access to a portion of the memory. Accessing memory that is local to a processor or core is faster than accessing remote memory, which is farther away in terms of physical distance.

NUMA[23] zones represent the different regions of memory within a single node, with varying access latencies depending on whether the memory is local or remote to a particular processor or core. Managing NuMA zones effectively is important for optimizing memory access performance in NUMA systems, as minimizing the impact of non-uniform memory access latencies can improve overall system performance and efficiency. Strategies such as memory affinity, thread placement, and data locality optimization are commonly used to optimize memory access patterns and reduce the effects of NUMA zones in HPC applications.

The peaks in energy consumption as shown in the 6.1 at thread counts of 12 and 16 could indeed be attributed to NUMA effects.

When the number of threads is increased, it can lead to increased memory access patterns, which may not be uniformly distributed across all memory banks. In NUMA architectures, different memory banks may have different access latencies, resulting in non-uniform memory access patterns.

As the number of threads increases, the memory access patterns may change, causing some threads to access memory in different NUMA zones with higher latency. This can lead to increased contention for memory resources and higher energy consumption, especially at thread counts where the memory access patterns result in increased NUMA effects, such as at 12 and 16 threads.

To mitigate the impact of NUMA effects on energy consumption, optimizing memory access patterns, thread affinity, and workload distribution can be beneficial. Additionally, profiling and analyzing memory access patterns can help identify and address potential bottlenecks associated with NUMA effects in HPC applications. NUMA affect was only noted significantly for LU-A and SP-A applications over miniHPC. The nodes in miniHPC have lesser cores (20) than in sciCORE (128) would be one of the reasons, why the NUMA affect is more prevalent in the former.

6.6 My carbon footprint

As per the figure 6.2, from 1.Aug 2023 to 2.April 2024, my utilization of the miniHPC cluster has resulted in a carbon footprint of 15 kgCO₂ (kilogram of carbon dioxide equivalent per kilogram), equivalent to 16 tree-months or driving 87 km. Despite achieving completion for 742 out of 980 jobs, 24.3% job failure rate indicates inefficiency, leading to the emission of 1 kgCO₂e. Furthermore, a significant overestimation of memory needs approximately 10 times higher than necessary, suggests a potential reduction in carbon emissions by requesting only required resources. Energy usage is dominated by CPU utilization (55%), followed by data center overheads (40%), while GPUs remain unused. With a carbon intensity of 467 gCO₂e/kWh, the total energy consumption stands at 32.46 kWh. This period saw 8 days and 7 hours of CPU usage out of 2 days and 6 hours of wall-clock time, with a total memory request of 69,874 GB. This comprehensive analysis underscores opportunities for enhancing efficiency and reducing environmental impact in future HPC cluster usage.

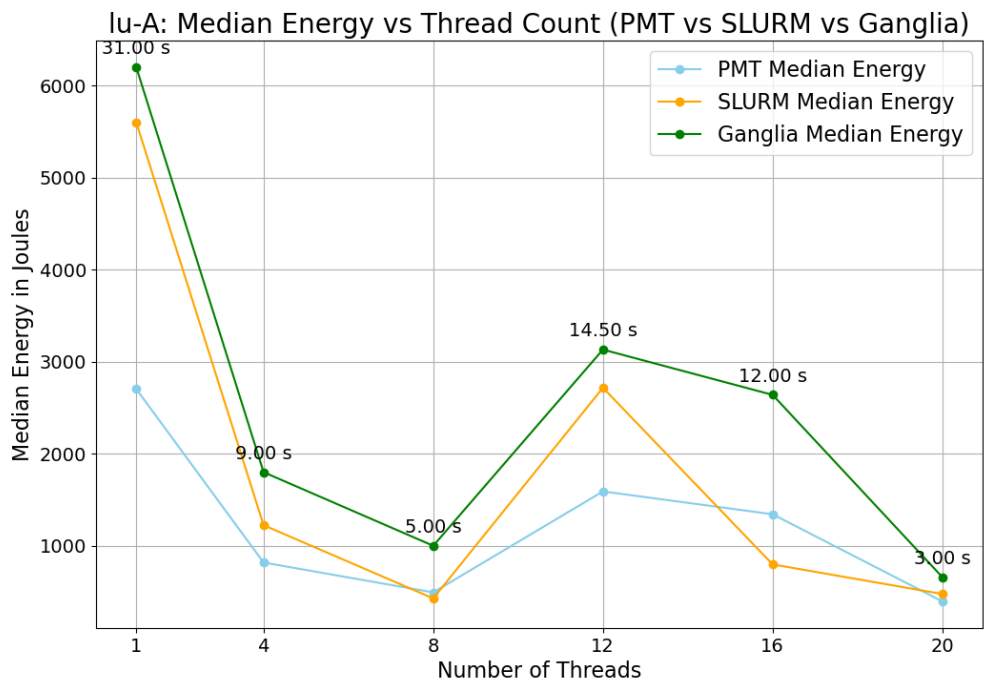


Figure 6.1: NUMA affect on miniHPC

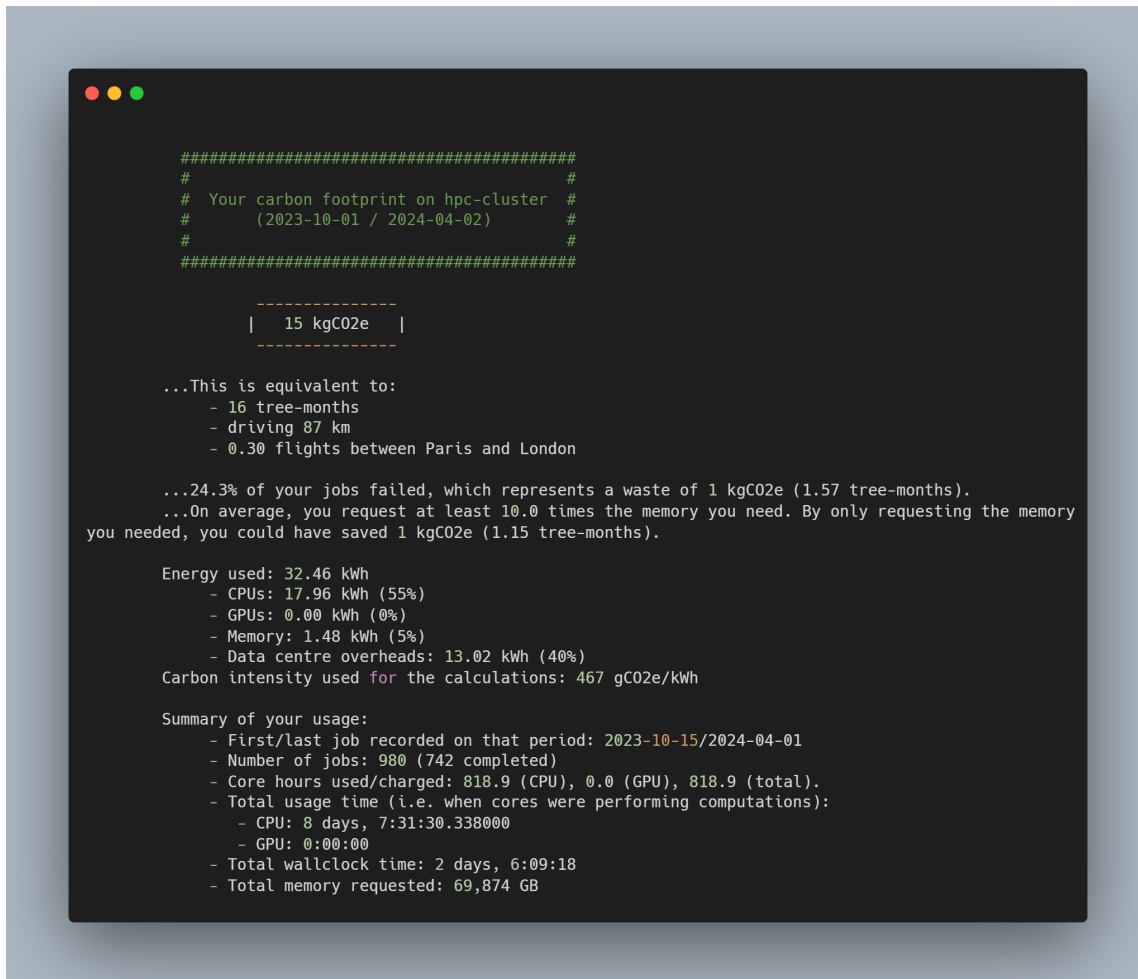


Figure 6.2: My carbon footprint for this thesis on miniHPC Cluster

Chapter 7

Conclusion

7.1 Contributions from the work

In this thesis, we have established an infrastructure for integrated energy monitoring on High-Performance Computing (HPC) systems. Using various software solutions such as PMT, LIKWID, Ganglia, and others, a bash wrapper script has been written to report energy consumption in an integrated and comparative manner.

Our setup underwent rigorous testing using multiple applications from the NAS benchmark suite, encompassing different programming paradigms and CPU frequencies across diverse HPC systems, including sciCORE and miniHPC. A comparative analysis employing plots, graphs, and tables provided valuable insights into energy consumption patterns.

Throughout this endeavor, several vital lessons were learned. We encountered challenges such as SLURM data unavailability inside jobs and the requirement for extensive permissions for specific software like LIKWID. Additionally, inconsistencies were observed in reporting, notably with PMT consistently reporting lower values.

Despite these challenges, our data analysis revealed significant findings, we collected the data across node, job, and application levels. Notably, we observed that increasing the number of ranks/threads led to reduced energy consumption, with some NUMA effects as corner cases and higher CPU frequencies were associated with lower energy usage.

In conclusion, our integrated energy monitoring solution offers users a comprehensive tool-set for obtaining detailed insights into energy consumption across various levels (node, job, application) in HPC environments. By leveraging this infrastructure, users can identify optimal rank/thread combinations, pinpoint energy-wasting applications, and ultimately work towards reducing their carbon footprint by optimizing their experiments. This framework represents a significant step towards promoting energy efficiency and sustainability in HPC research and practice.

7.2 Future Work

While the integrated energy monitoring infrastructure presented in this thesis represents a significant advancement in understanding energy consumption patterns in HPC systems, several avenues for future research and improvement remain to be explored. In this section, potential directions for

future work have been outlined based on the limitations and opportunities identified during this research.

7.2.1 Job Co-location Strategies

Job co-location is the simultaneous execution of multiple compute tasks on the same hardware resources. It presents an opportunity to improve resource utilization and energy efficiency in HPC clusters. Future work could investigate novel job scheduling algorithms and co-location strategies that intelligently group compatible tasks based on their resource requirements, communication patterns, and affinity to minimize contention and resource fragmentation. By strategically co-locating jobs, we can reduce idle resource time and mitigate energy wastage associated with underutilized hardware, thereby enhancing overall system efficiency. Develop algorithms or heuristics to determine which jobs should be co-located on the same node based on various factors such as resource requirements, communication patterns, and workload characteristics. This decision-making process could consider factors like job runtime, resource utilization, and potential energy savings. The following combination of factors could be taken into account

- **Compute Job + Compute Job:** Investigate whether co-locating compute-intensive jobs on the same node leads to increased energy consumption due to higher CPU utilization or whether these jobs can efficiently share resources and benefit from parallel execution. Analyze the impact of workload characteristics such as CPU usage patterns and memory access patterns on energy consumption.
- **Compute Job + Memory Job:** Explore the energy implications of co-locating compute-intensive jobs with memory-intensive jobs. Determine whether memory-bound jobs can benefit from the presence of compute-bound jobs on the same node by leveraging idle CPU resources or whether the competition for memory bandwidth leads to increased energy consumption.
- **Memory Job + Memory Job:** Evaluate the energy efficiency of co-locating memory-bound jobs on the same node. Analyze the impact of memory contention and cache utilization on energy consumption and performance. Investigate whether memory-bound jobs can effectively share memory resources without significantly impacting energy consumption.
- **Compute Job + Network Job:** Assess the energy implications of co-locating compute-intensive jobs with network-bound jobs. Analyze the impact of network communication overhead on energy consumption and performance. Investigate whether co-location leads to energy savings by reducing data transfer latency or whether it increases energy consumption due to higher network utilization.
- **Other Combinations with I/O bound jobs:** Extend the analysis to other combinations of job types, such as I/O-bound jobs, to understand their energy implications when co-located on the same node. Investigate the trade-offs between resource sharing and contention in different co-location scenarios and their impact on energy efficiency.

Bibliography

- [1] scicore and university of basel, <https://scicore.unibas.ch/>.
- [2] Slurm workload manager- slurm power management guide.
- [3] Eishi Arima, A. Comprés, and Martin Schulz. *On the Convergence of Malleability and the HPC PowerStack: Exploiting Dynamism in Over-Provisioned and Power-Constrained HPC Systems*, pages 206–217. 01 2023.
- [4] Anonymous Author(s). Measuring application-level energy consumption of cosmological simulations. 2023.
- [5] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A case study of energy aware scheduling on supermuc. In *2014 Teratec HPC Forum*, pages 1–6. IEEE, 2014.
- [6] David H. Bailey. *NAS Parallel Benchmarks*, pages 1254–1259. Springer US, Boston, MA, 2011.
- [7] Rakesh Bhatnagar and Jayeshkumar Patel. Performance analysis of a grid monitoring system-ganglia. 3:362–365, 08 2013.
- [8] Paweł Bratek, Łukasz Szustak, Roman Wyrzykowski, and Tomasz Olas. Reducing energy consumption using heterogeneous voltage frequency scaling of data-parallel applications for multicore systems. *Journal of Parallel and Distributed Computing*, 175, 05 2023.
- [9] Daniele Cesarini, Andrea Bartolini, Carlo Cavazzoni Andrea Borghesi, Mathieu Luisier, and Luca Benini. Countdown slack: a run-time library to reduce energy footprint in large-scale mpi application. 2018.
- [10] Ghislain Landry Tsafack Chetsa, Laurent Lefevre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa. A runtime framework for energy efficient hpc systems without a priori knowledge of applications. In *International Conference on Parallel and Distributed Systems*, pages 1–10. IEEE, 2012.
- [11] Ghislain Landry Tsafack Chetsa, Laurent Lefevre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa. Exploiting performance counters to predict and improve energy performance of hpc systems. *Future Generation Computer Systems*, 36:287–298, 2014.

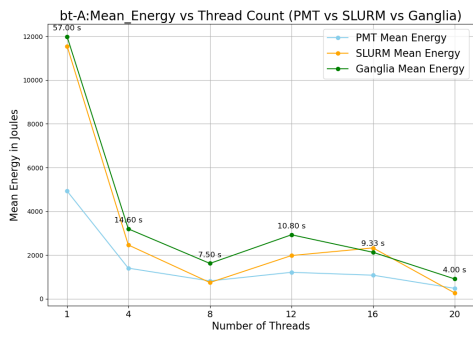
- [12] Jacob Combs, Jolie Nazon, Rachelle Thysell, Lowell Olson Fabian Santiago, Matthew Hardwick, Suzanne Rivoire, Chung-Hsing Hsu, and Stephen W. Poole. Power signatures of high-performance computing workloads. *2014 Energy Efficient Supercomputing Workshop*, pages 70–78, 2014.
- [13] Christian Conficoni, Andrea Bartolini, Andrea Tilli, Giampietro Tecchiolli, and Luca Benini. Energy-aware cooling for hot-water cooled supercomputers. 2014.
- [14] Julita Corbalan and Luigi Brochard. Ear: Energy management framework for supercomputers. 2018.
- [15] Stefano Corda, Bram Veenboer, and Emma Tolley. Pmt: Power measurement toolkit, 10 2022.
- [16] Neha Gholkar, Frank Mueller, Barry Rountree, and Aniruddha Marathe. Pshifter: feedback-based dynamic power shifting within hpc jobs for performance. In *2018 IEEE International Conference on High-Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12. IEEE, 2018.
- [17] Jason Grealey, Loïc Lannelongue, Woei-Yuh Saw, Jonathan Marten, Guillaume Méric, Sergio Ruiz-Carmona, and Michael Inouye. The carbon footprint of bioinformatics. *Molecular Biology and Evolution*, 39, 02 2022.
- [18] Thomas Gruber, Jan Eitzinger, Georg Hager, and Gerhard Wellein. Likwid monitoring stack: A flexible framework enabling job specific performance monitoring for the masses. pages 781–784, 09 2017.
- [19] Thomas Gruber, Jan Eitzinger, Georg Hager, and Gerhard Wellein. Likwid, November 2023.
- [20] Connor Imes, Steven Hofmeyr, and Henry Hofmann. Energy-efficient application resource scheduling using machine learning classifiers. pages 1–12. IEEE, 2018.
- [21] Richard Kavanagh and Karim Djemame. Rapid and accurate energy models through calibration with ipmi and rapl. *Concurrency and Computation: Practice and Experience*, 31(13):e5124, 2019. e5124 cpe.5124.
- [22] Gregory A Koenig, Matthias Maiterth, Siddhartha Jana, Natalie Bates, Kevin Pedretti, Milos Puzovic, Andrea Borghesi, Andrea Bartolini, and David Montoya. Energy and power-aware job scheduling and resource management: Global survey—initial analysis. In *2018 IEEE International Conference on High-Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12. IEEE, 2018.
- [23] Christoph Lameter. Numa (non-uniform memory access): An overview. *Queue*, 11, 07 2013.
- [24] Loïc Lannelongue. Carbon footprint: the (not so) hidden cost of high-performance computing. *ITNOW*, 63:12–13, 01 2022.
- [25] Loïc Lannelongue, Hans-Erik Aronson, Alex Bateman, Ewan Birney, Talia Caplan, Martin Juckes, Jo Mcentyre, Andrew Morris, Gerry Reilly, and Michael Inouye. Greener principles for environmentally sustainable computational science. *Nature Computational Science*, 3:514–521, 06 2023.

- [26] Loïc Lannelongue, Jason Grealey, Alex Bateman, and Michael Inouye. Ten simple rules to make your computing more environmentally sustainable. *PLOS Computational Biology*, 17:e1009324, 09 2021.
- [27] Loïc Lannelongue, Jason Grealey, and Michael Inouye. Green algorithms: Quantifying the carbon footprint of computation. *Advanced Science*, 8, 05 2021.
- [28] Loïc Lannelongue and Michael Inouye. Carbon footprint estimation for computational research. *Nature Reviews Methods Primers*, 3, 02 2023.
- [29] Matthias Maiterth, Asma Al-Rawi, Fede Ardanaz, Brandon Baker, Chris Cantalupo, Jonathan Eastep, Brad Geltz, Diana Guttman, Siddhartha Jana, Fuat Keceli, Kelly Livingston, et al. Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions. In *2018 IEEE International Conference on High-Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12. IEEE, 2018.
- [30] Matt Massie. *Monitoring with Ganglia*. O’Reilly Media, Sebastopol, CA, October 2012.
- [31] Matthew Massie, Brent Chun, and David Culler. The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, 30:817–840, 07 2004.
- [32] Gence Ozer, Sarthak Garg, Neda Davoudi, Gabrielle Poerwawinata, Matthias Maiterth, Alessio Netti, and Daniele Tafani. Towards a predictive energy model for hpc runtime systems using supervised learning. 08 2019.
- [33] Prometheus. Prometheus-monitoring system and time series database.
- [34] Bjorn Rabenstein and Julius Volz. Prometheus: A next-generation monitoring system (talk). Dublin, May 2015. USENIX Association.
- [35] Rrze-Hpc. Rrze-hpc/likwid: Performance monitoring and benchmarking suite.
- [36] Issa Saba, Eishi Arima, Dai Liu, and Martin Schulz. *Orchestrated Co-scheduling, Resource Partitioning, and Power Capping on CPU-GPU Heterogeneous Systems via Machine Learning*, pages 51–67. 12 2022.
- [37] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Predicting the energy and power consumption of strong and weak scaling hpc applications. *Supercomputing Frontiers and Innovations*, pages 20–41, 2014.
- [38] G. Hager T. Gruber, J. Eitzinger and G. Wellein. Likwid 5: Lightweight performance tools. 2023.
- [39] N. Lachiche T. Jakobsche and F. M. Ciorba. Investigating hpc job resource requests and job efficiency reporting. 2023.
- [40] Robert Tracey, Lan Hoang, Felix Soubelet, and Vadim Elisseev. *AI-Driven Holistic Approach to Energy Efficient HPC*, pages 267–279. 10 2020.
- [41] Laurens Versluis, Mehmet Cetin, Caspar Greeven, Kristian Laursen, Damian Podareanu, Valeriu Codreanu, Alexandru Uta, and Alexandru Iosup. A holistic analysis of datacenter operations: Resource usage, energy, and workload characterization – extended technical report. 07 2021.

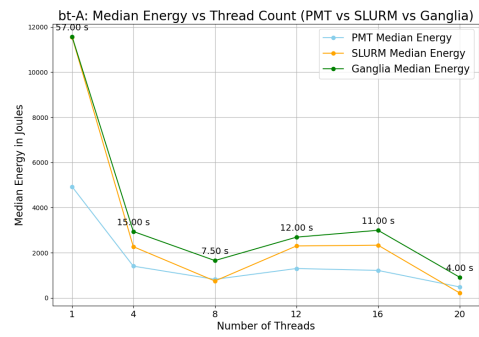
- [42] Jan Weglarz, Pawel Czarnul, Jerzy Proficz, and Adam Krzywaniak. Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, 2019.
- [43] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [44] Jumie Yuventi and Roshan Mehdizadeh. A critical analysis of power usage effectiveness and its use in communicating data center energy consumption. *Energy and Buildings*, 61:90–94, 2013.

Chapter 8

Appendix

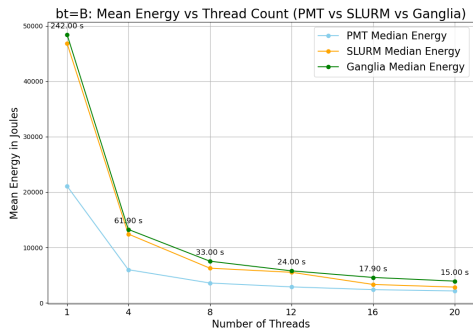


(a) Mean

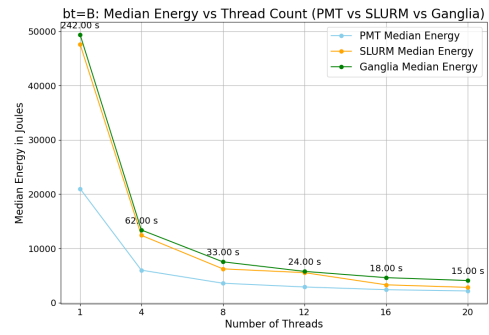


(b) Median

Figure 8.1: BT for Class A, OpenMP

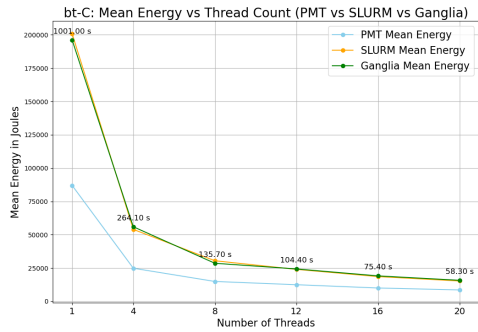


(a) Mean

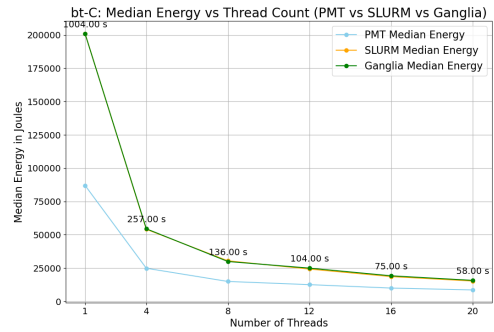


(b) Median

Figure 8.2: BT for Class B, OpenMP

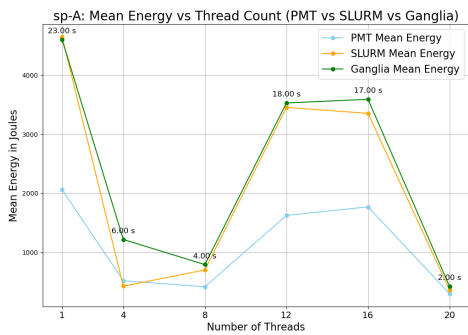


(a) Mean

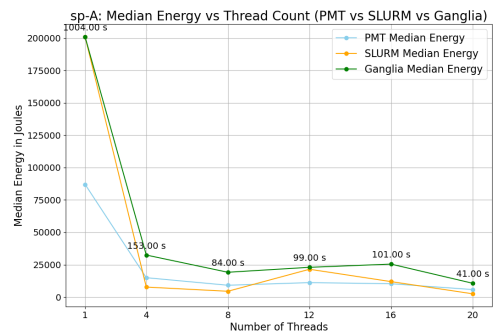


(b) Median

Figure 8.3: BT for Class C, OpenMP

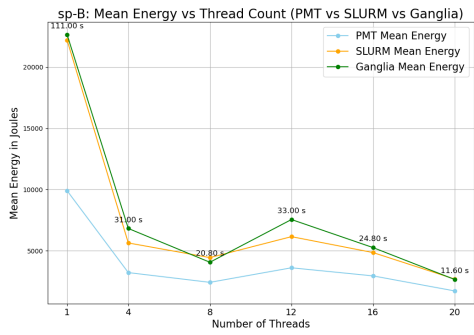


(a) Mean

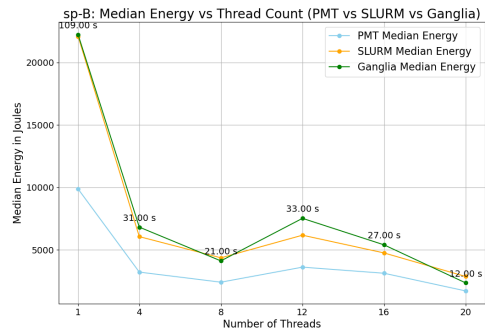


(b) Median

Figure 8.4: SP for Class A, OpenMP

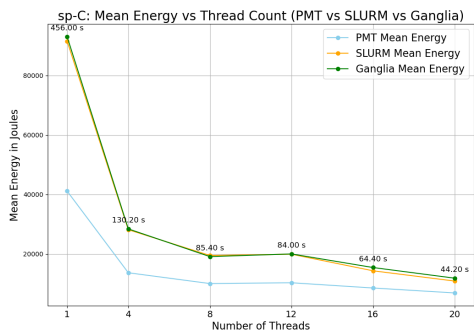


(a) Mean

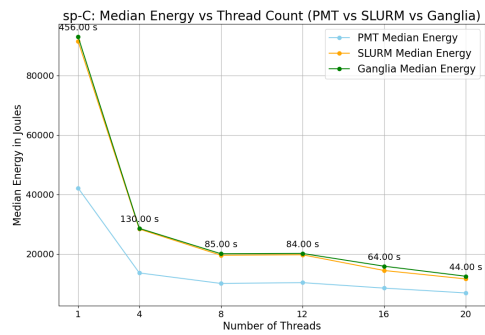


(b) Median

Figure 8.5: SP for Class B, OpenMP

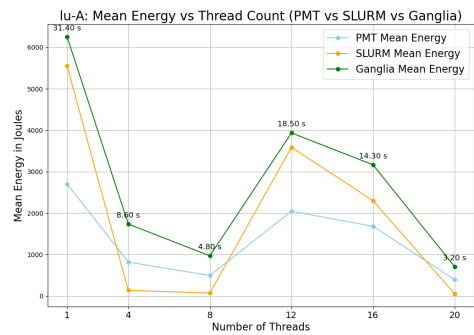


(a) Mean

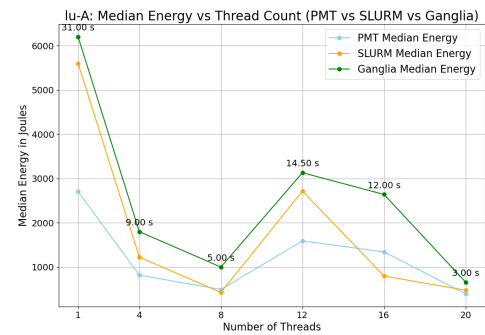


(b) Median

Figure 8.6: SP for Class C, OpenMP

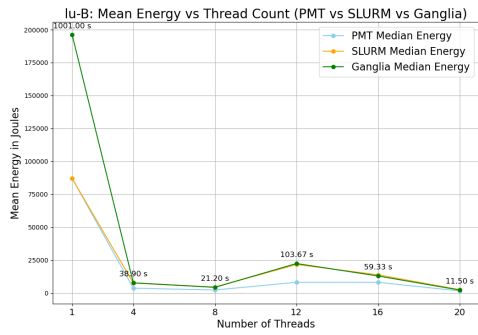


(a) Mean

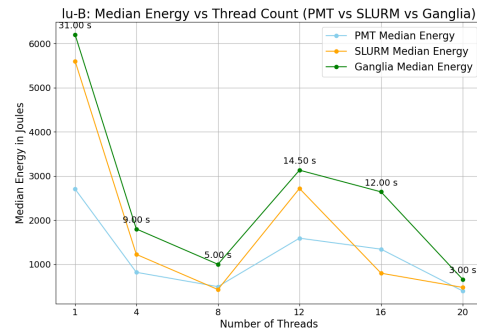


(b) Median

Figure 8.7: LU for Class A, OpenMP

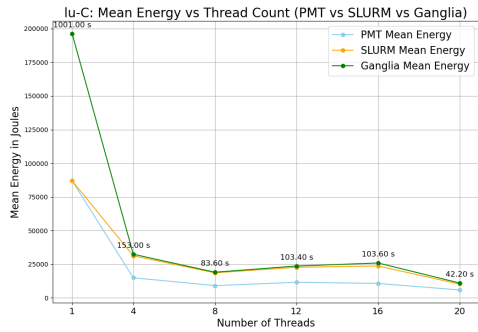


(a) Mean

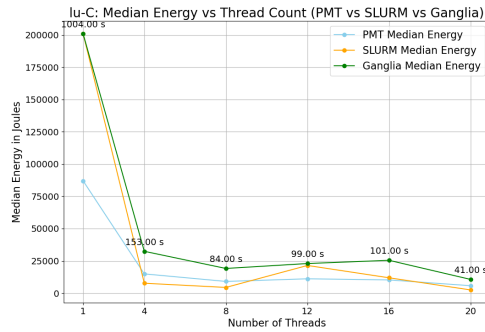


(b) Median

Figure 8.8: LU for Class B, OpenMP

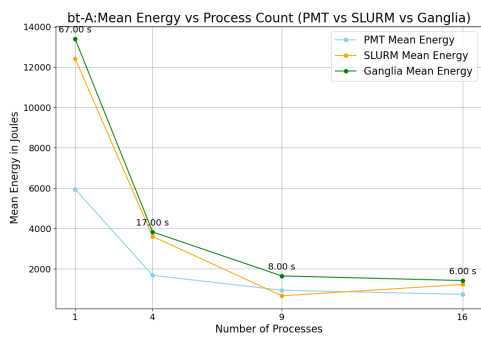


(a) Mean

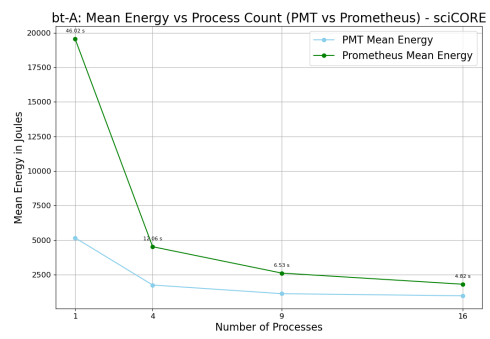


(b) Median

Figure 8.9: LU for Class C, OpenMP

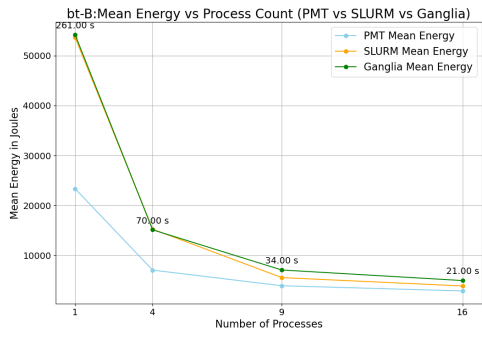


(a) miniHPC

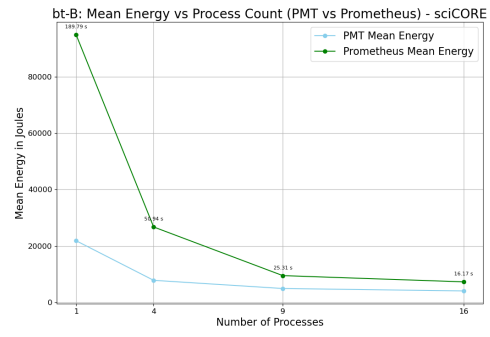


(b) sciCORE

Figure 8.10: BT for Class A, MPI

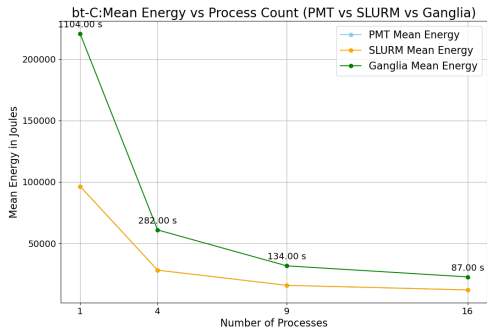


(a) miniHPC

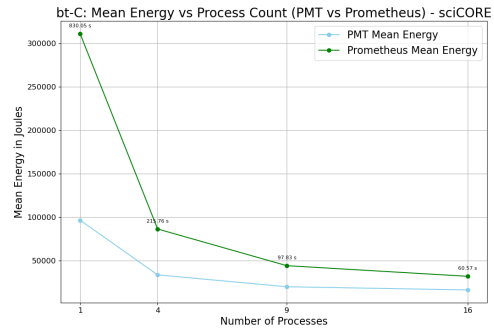


(b) sciCORE

Figure 8.11: BT for Class B, MPI

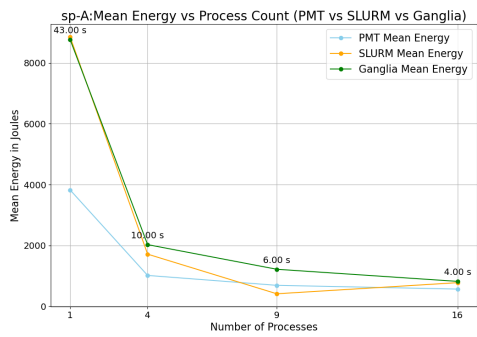


(a) miniHPC

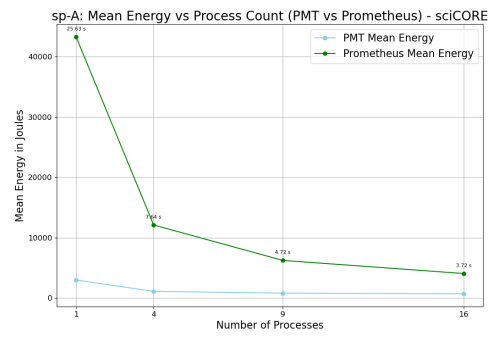


(b) sciCORE

Figure 8.12: BT for Class C, MPI

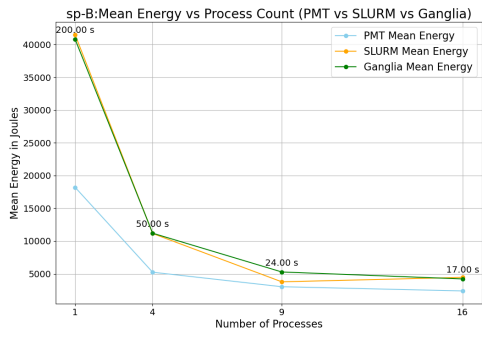


(a) miniHPC

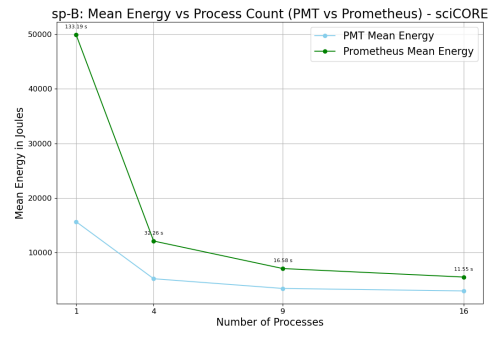


(b) sciCORE

Figure 8.13: SP for Class A, MPI

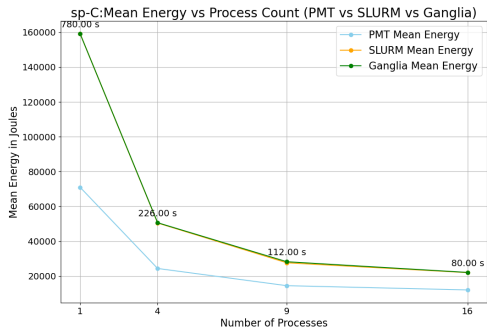


(a) miniHPC

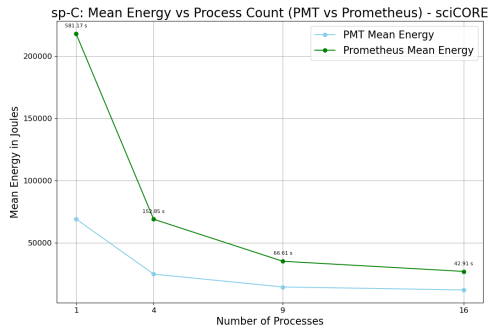


(b) sciCORE

Figure 8.14: SP for Class B, MPI

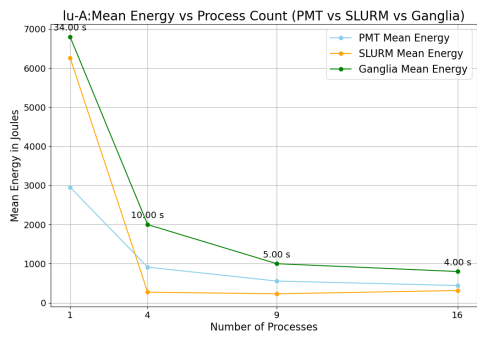


(a) miniHPC

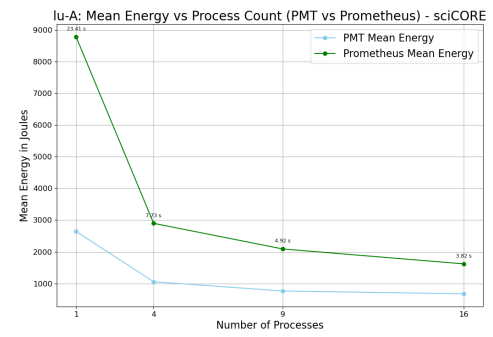


(b) sciCORE

Figure 8.15: SP for Class C, MPI

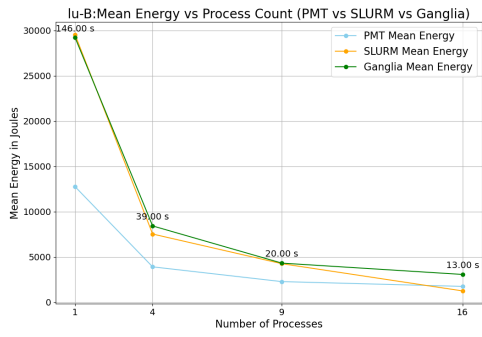


(a) miniHPC

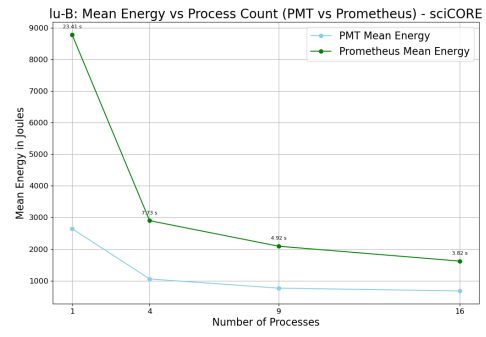


(b) sciCORE

Figure 8.16: LU for Class A, MPI

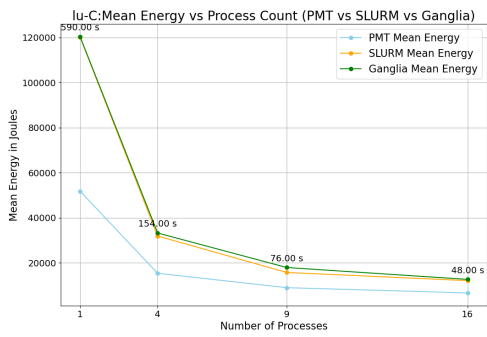


(a) miniHPC

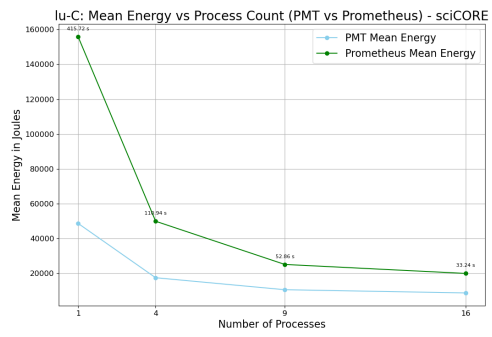


(b) sciCORE

Figure 8.17: LU for Class B, MPI

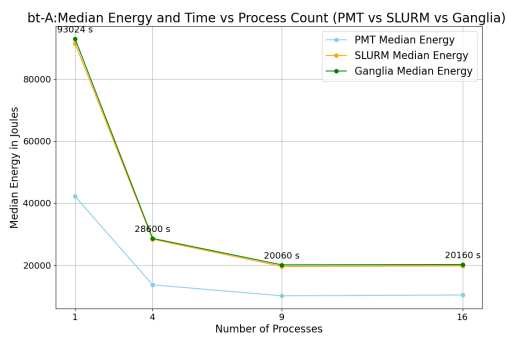


(a) miniHPC

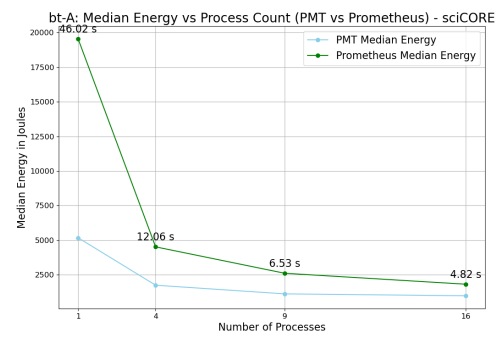


(b) sciCORE

Figure 8.18: LU for Class C, MPI

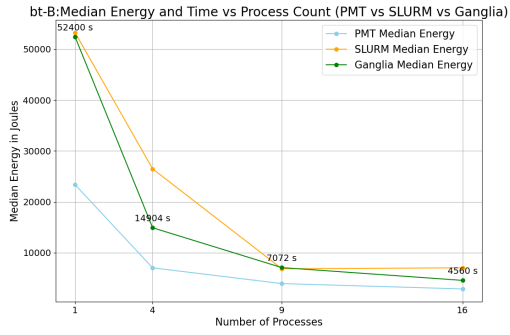


(a) miniHPC

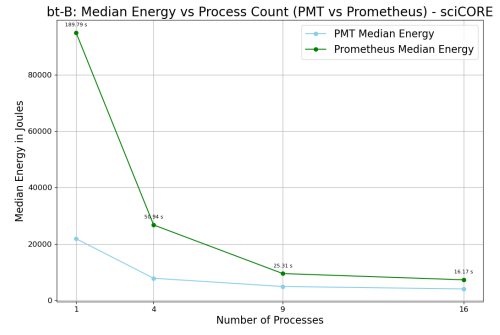


(b) sciCORE

Figure 8.19: BT for Class A, MPI

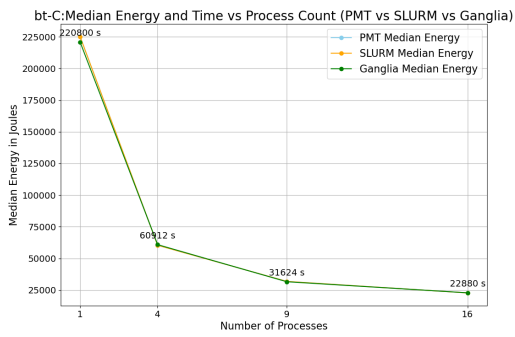


(a) miniHPC

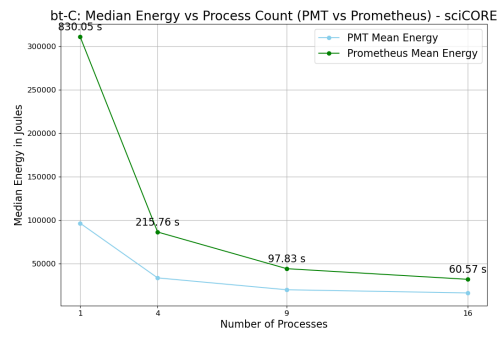


(b) sciCORE

Figure 8.20: BT for Class B, MPI

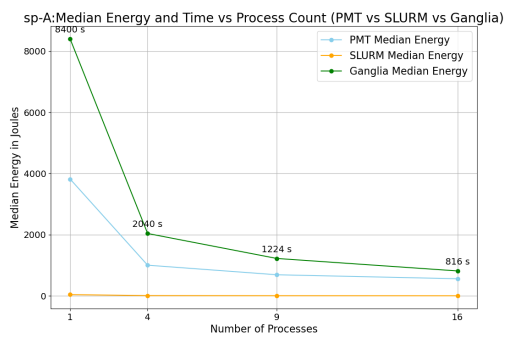


(a) miniHPC

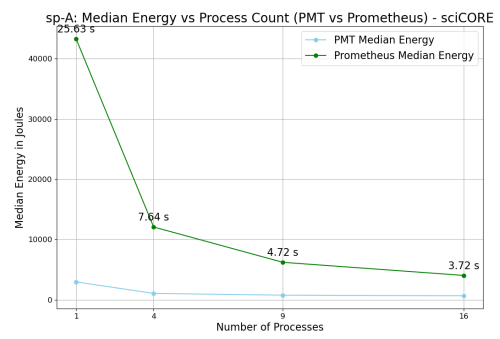


(b) sciCORE

Figure 8.21: BT for Class C, MPI

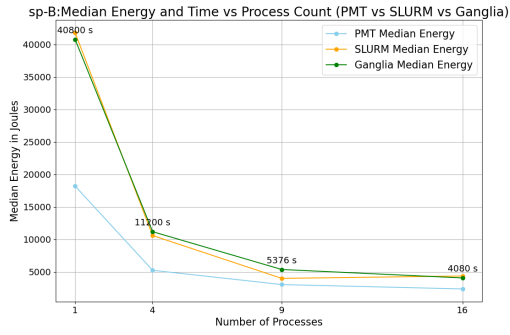


(a) miniHPC

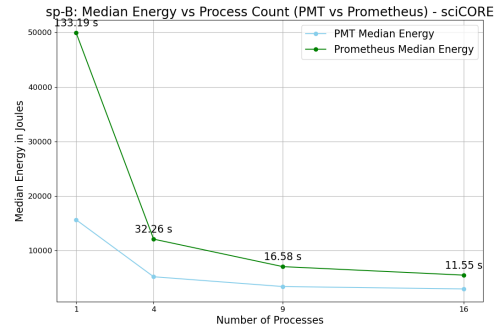


(b) sciCORE

Figure 8.22: SP for Class A, MPI

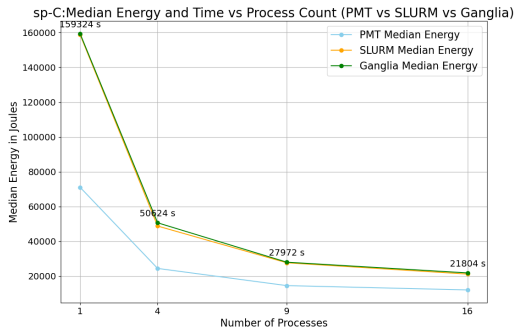


(a) miniHPC

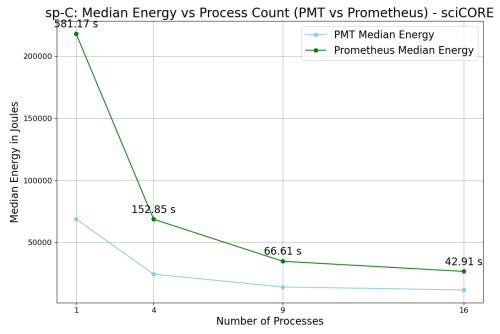


(b) sciCORE

Figure 8.23: SP for Class B, MPI

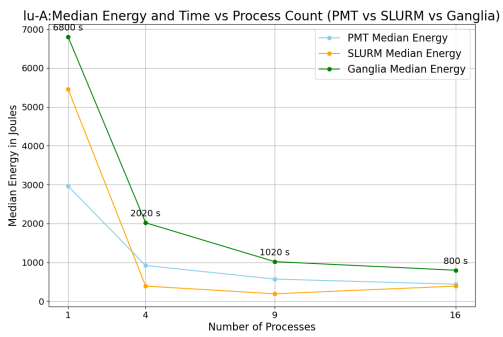


(a) miniHPC

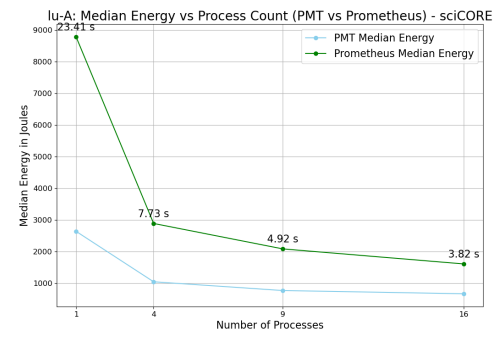


(b) sciCORE

Figure 8.24: SP for Class C, MPI

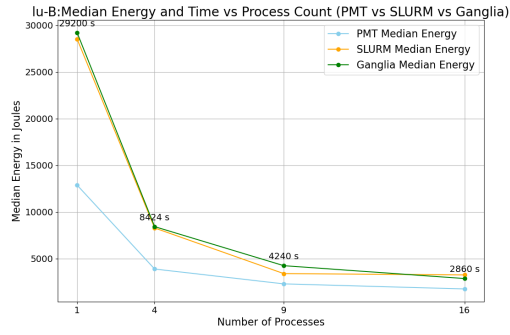


(a) miniHPC

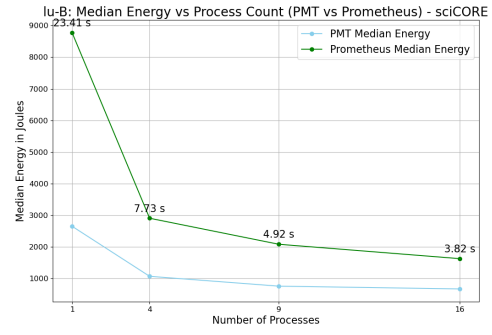


(b) sciCORE

Figure 8.25: LU for Class A, MPI

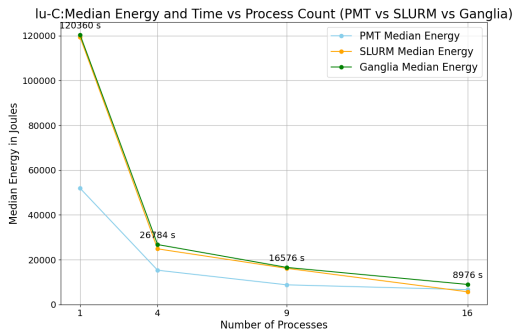


(a) miniHPC

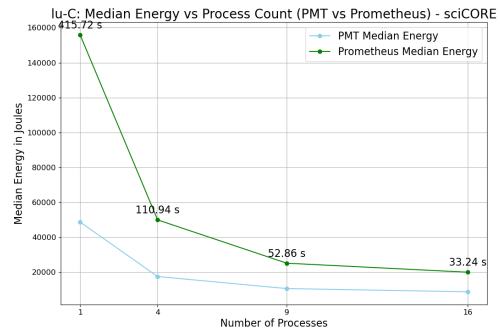


(b) sciCORE

Figure 8.26: LU for Class B, MPI

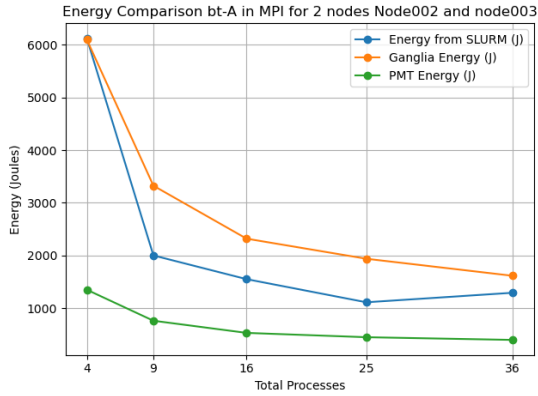


(a) miniHPC

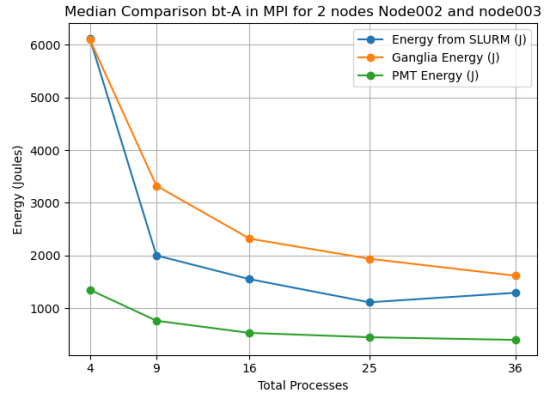


(b) sciCORE

Figure 8.27: LU for Class C, MPI

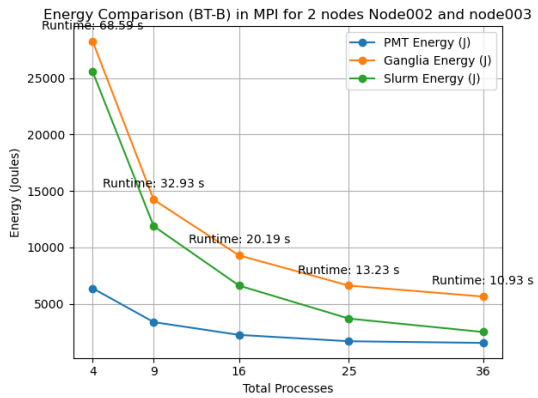


(a) Mean

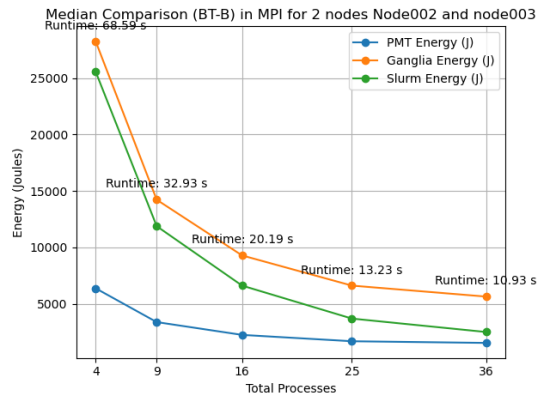


(b) Median

Figure 8.28: BT for Class a, MPI on 2 nodes node002 and node003

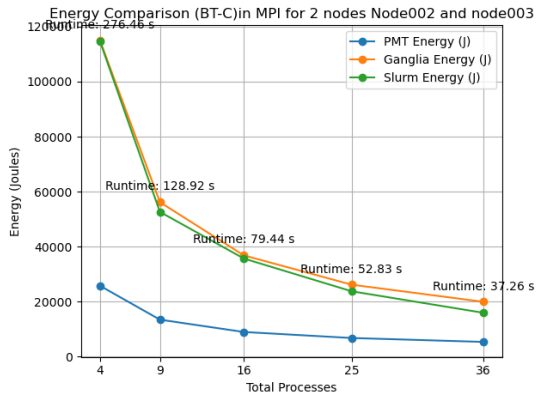


(a) Mean

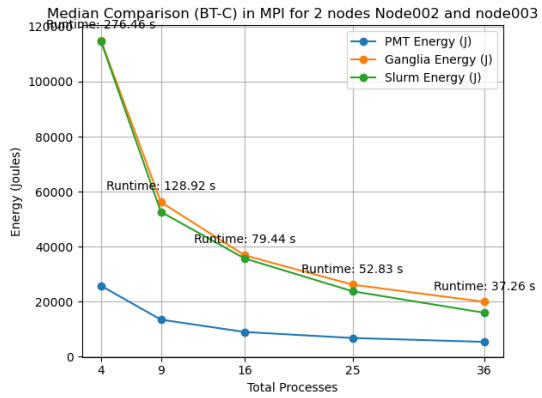


(b) Median

Figure 8.29: BT for Class b, MPI on 2 nodes node002 and node003

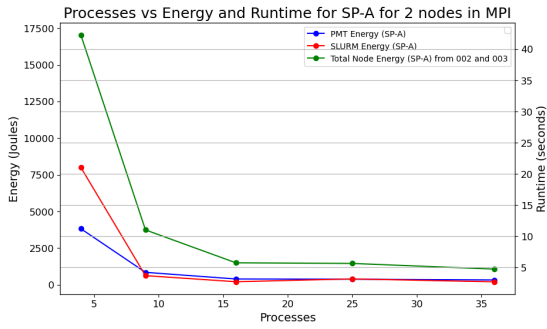


(a) Mean

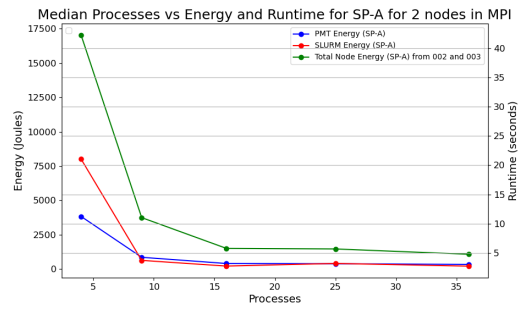


(b) Median

Figure 8.30: BT for Class c, MPI on 2 nodes node002 and node003

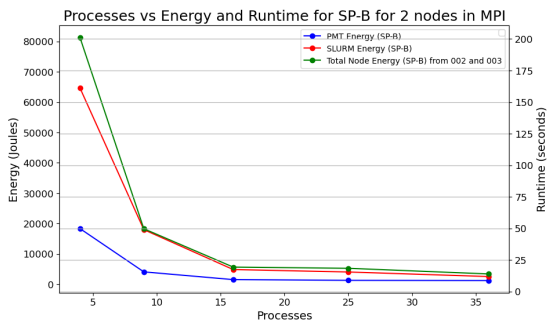


(a) Mean

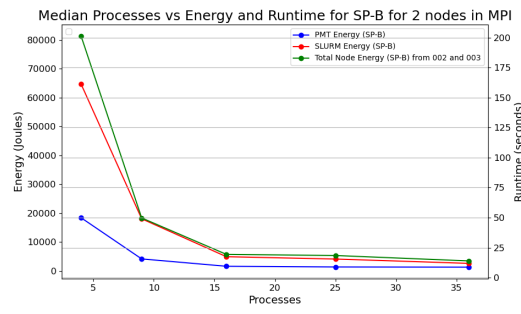


(b) Median

Figure 8.31: SP for Class a, MPI on 2 nodes node002 and node003

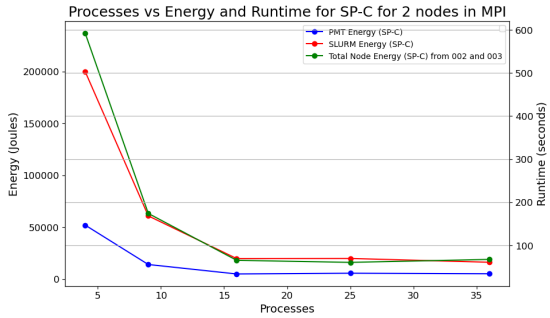


(a) Mean

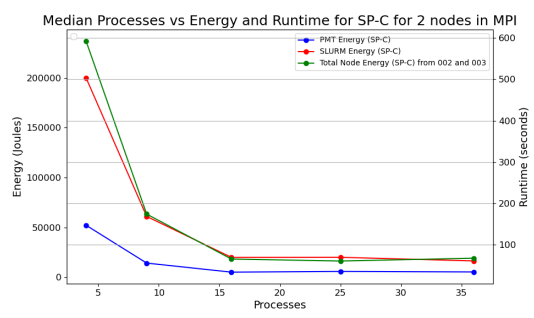


(b) Median

Figure 8.32: SP for Class b, MPI on 2 nodes node002 and node003

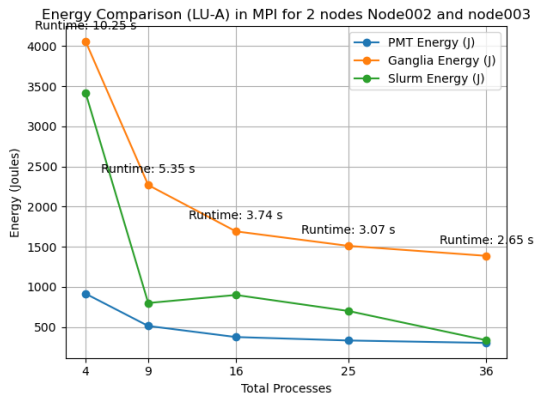


(a) Mean

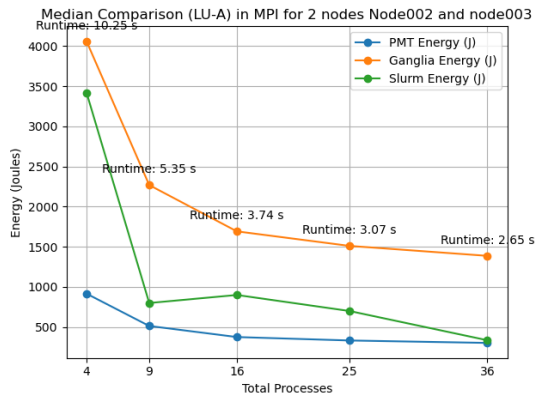


(b) Median

Figure 8.33: SP for Class c, MPI on 2 nodes node002 and node003

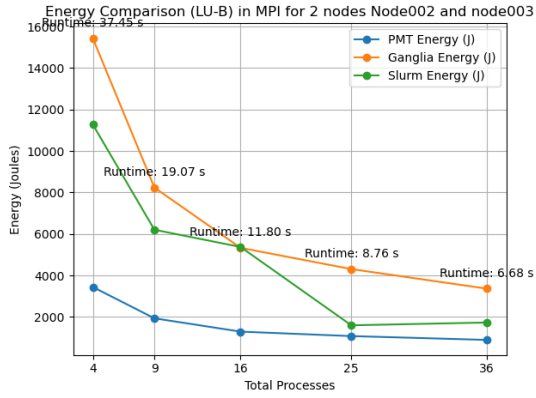


(a) Mean

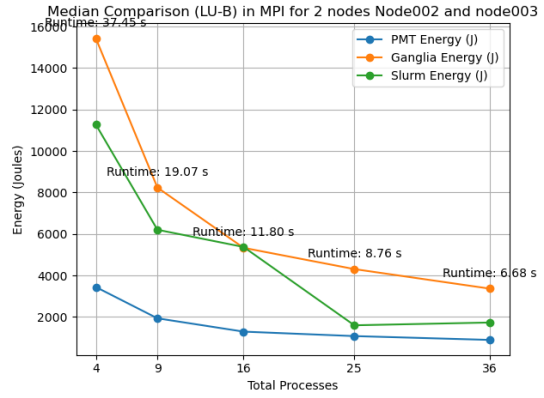


(b) Median

Figure 8.34: LU for Class a, MPI on 2 nodes node002 and node003

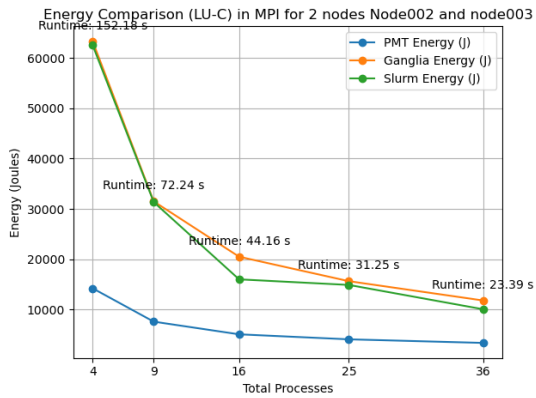


(a) Mean

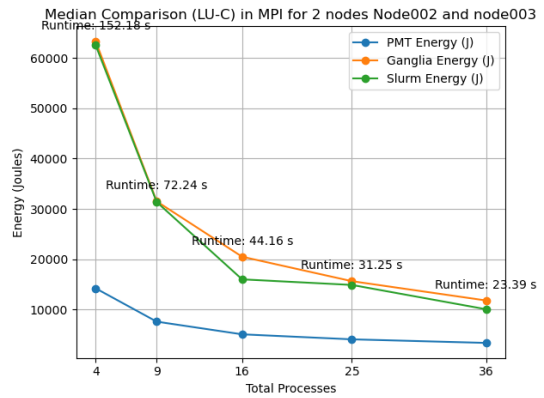


(b) Median

Figure 8.35: LU for Class b, MPI on 2 nodes node002 and node003



(a) Mean



(b) Median

Figure 8.36: LU for Class c, MPI on 2 nodes node002 and node003

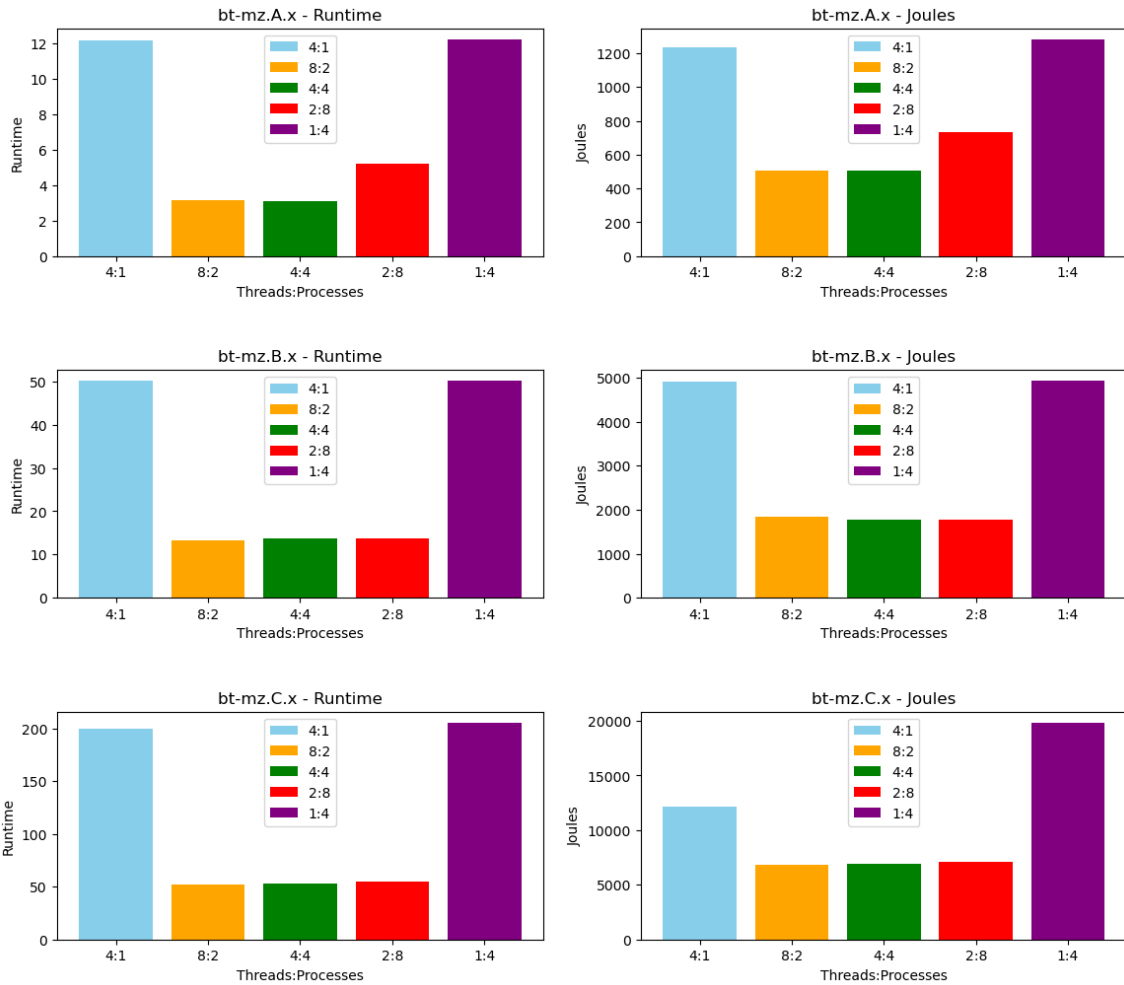


Figure 8.37: BT with OpenMP+MPI

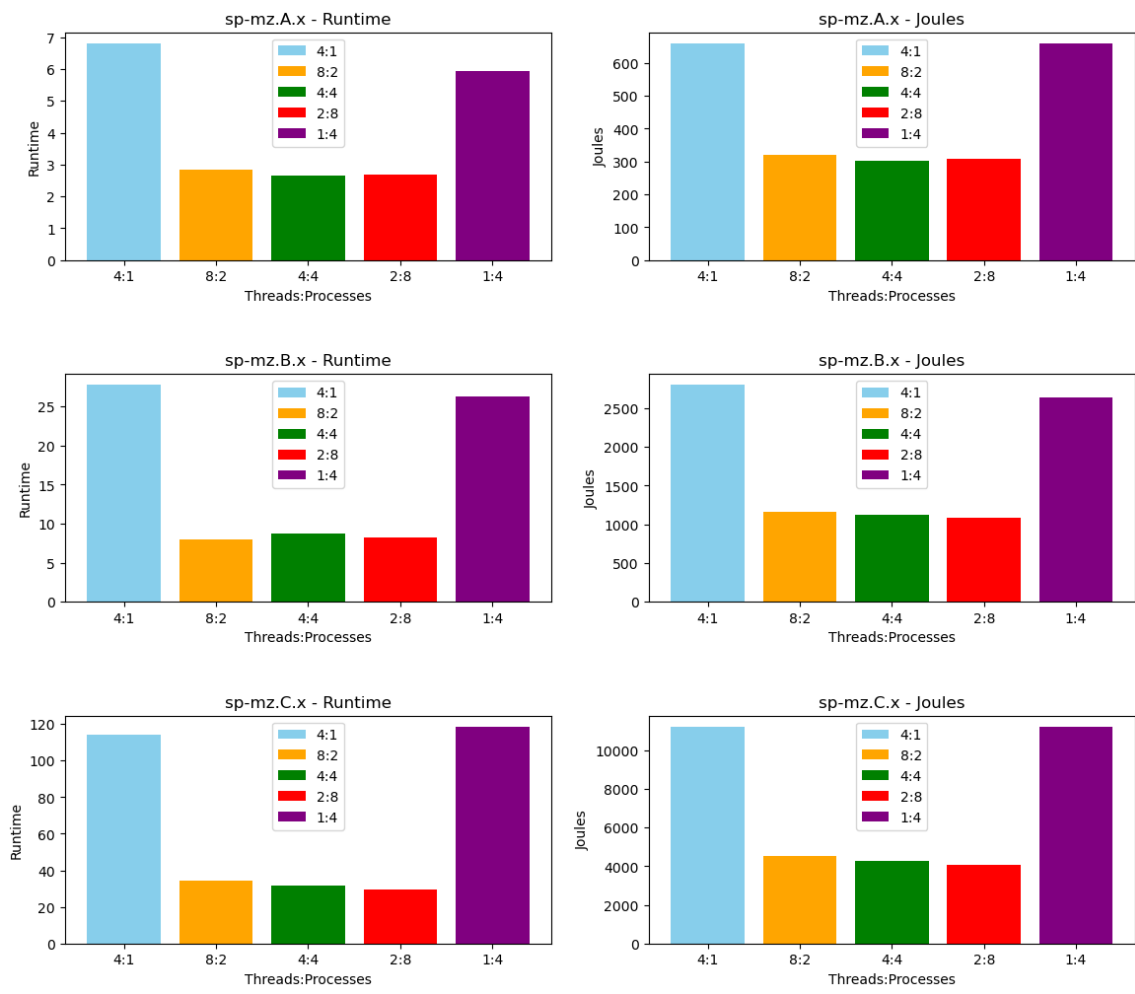


Figure 8.38: SP with OpenMP+MPI

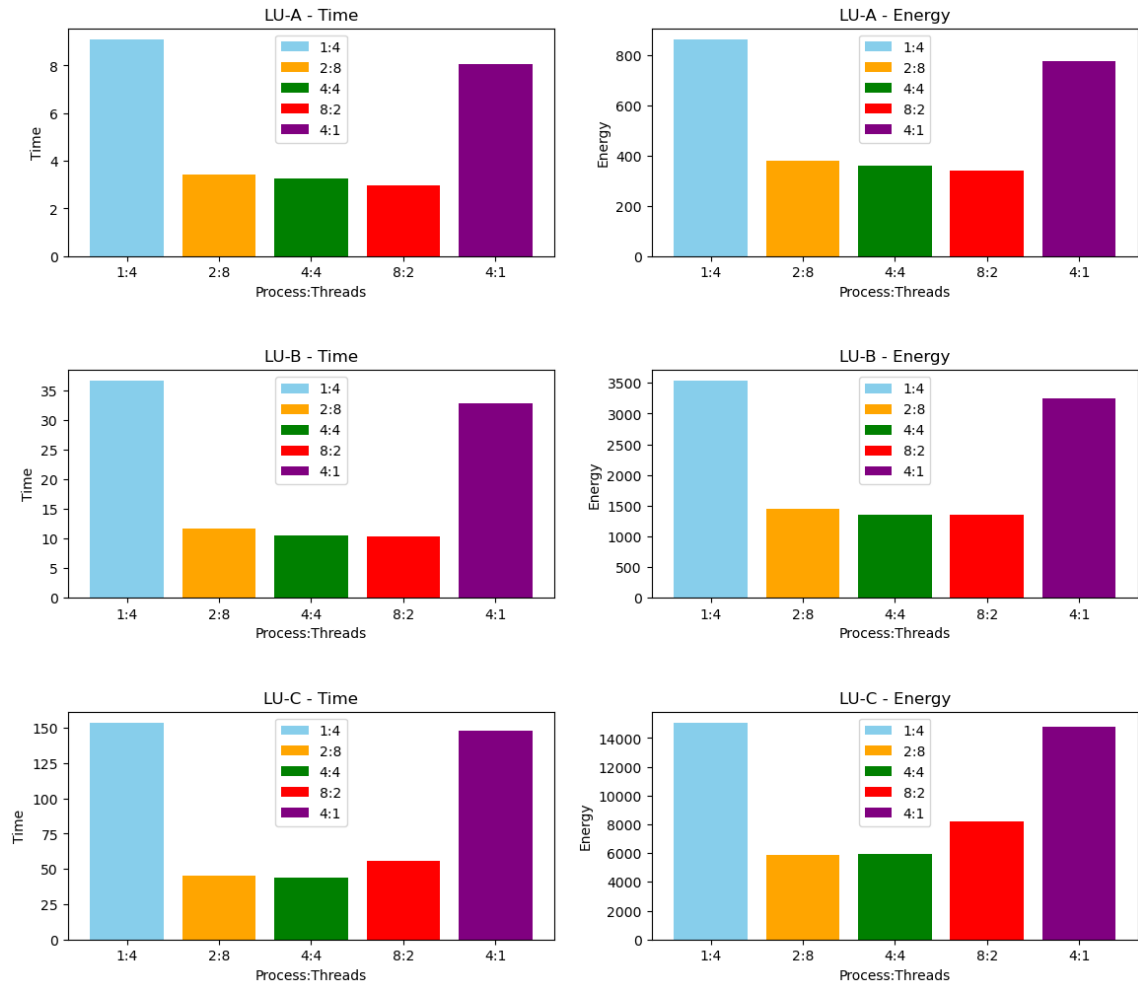


Figure 8.39: LU with OpenMP+MPI

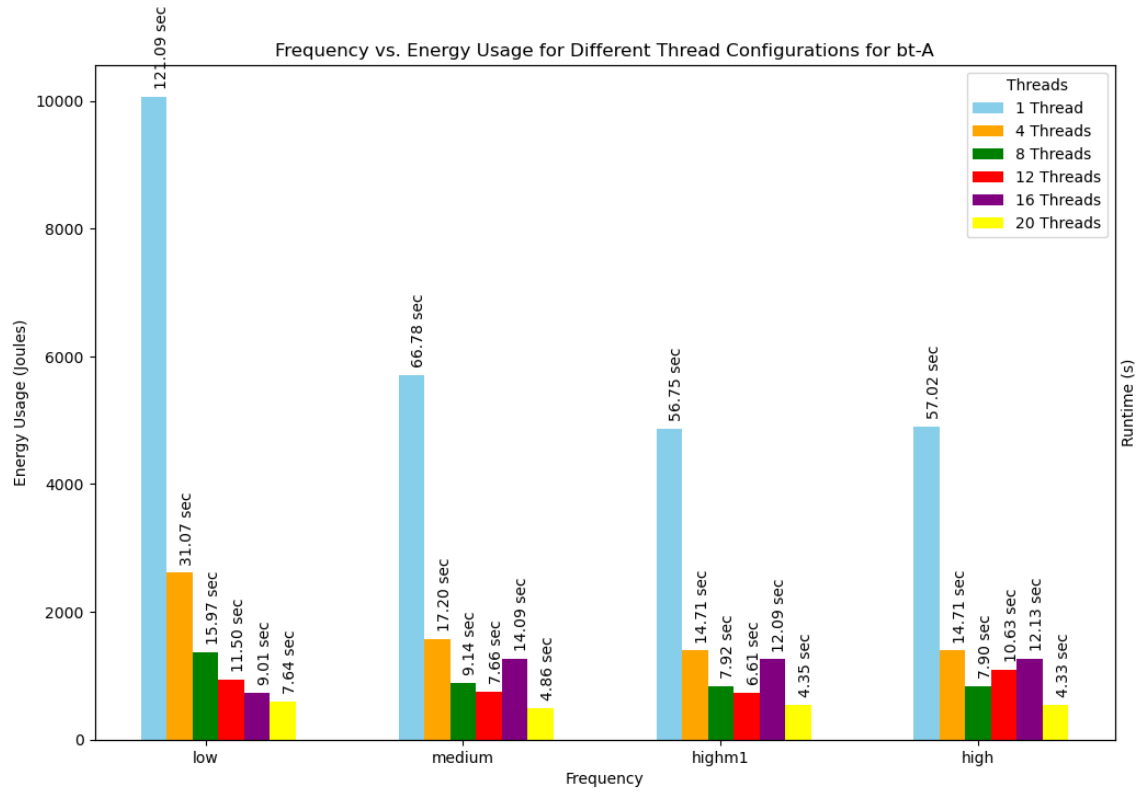


Figure 8.40: BT-A: Affect of CPU frequencies on the application level energy consumption for different thread count

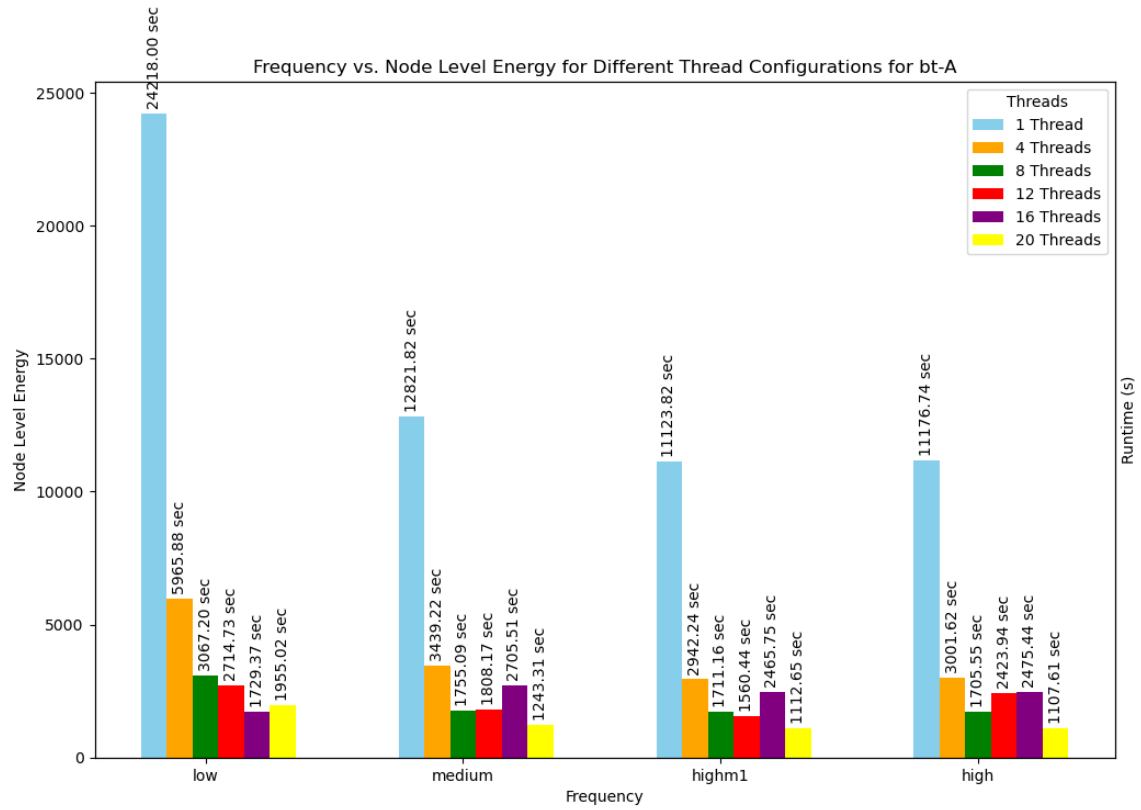


Figure 8.41: BT-A:Affect of CPU frequencies on the node level energy consumption for different thread count

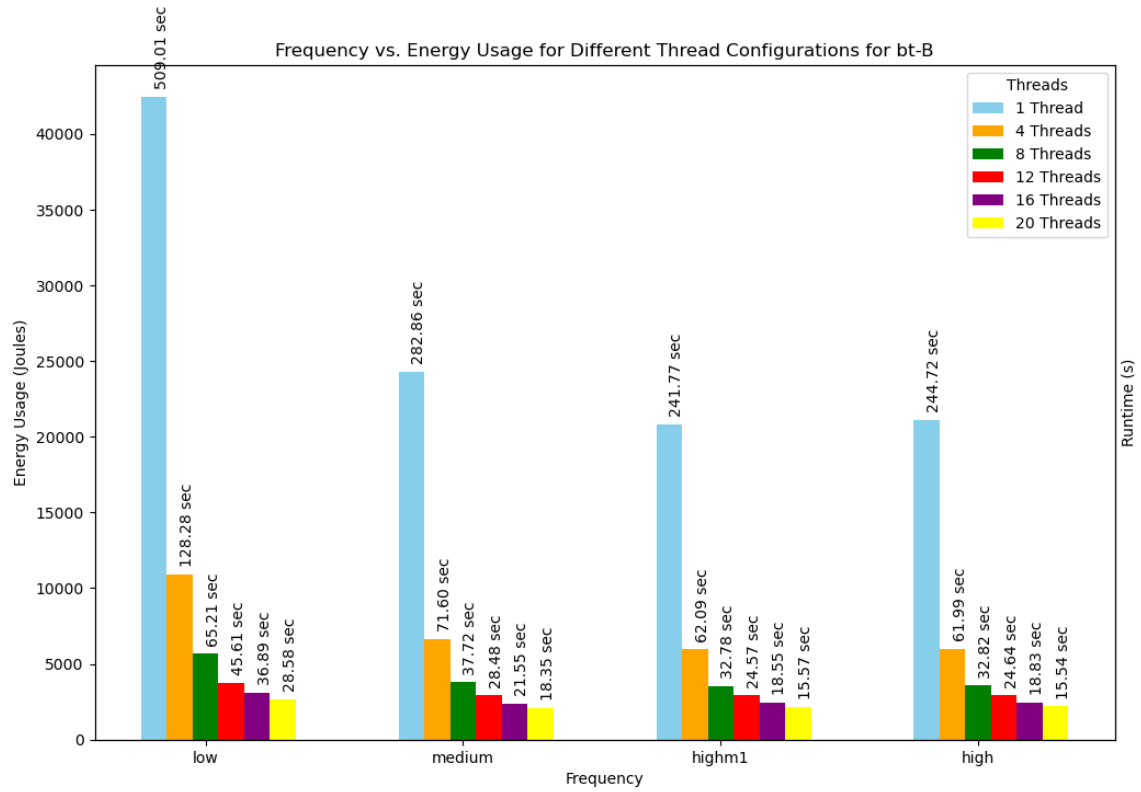


Figure 8.42: BT-B: Affect of CPU frequencies on the application level energy consumption for different thread count

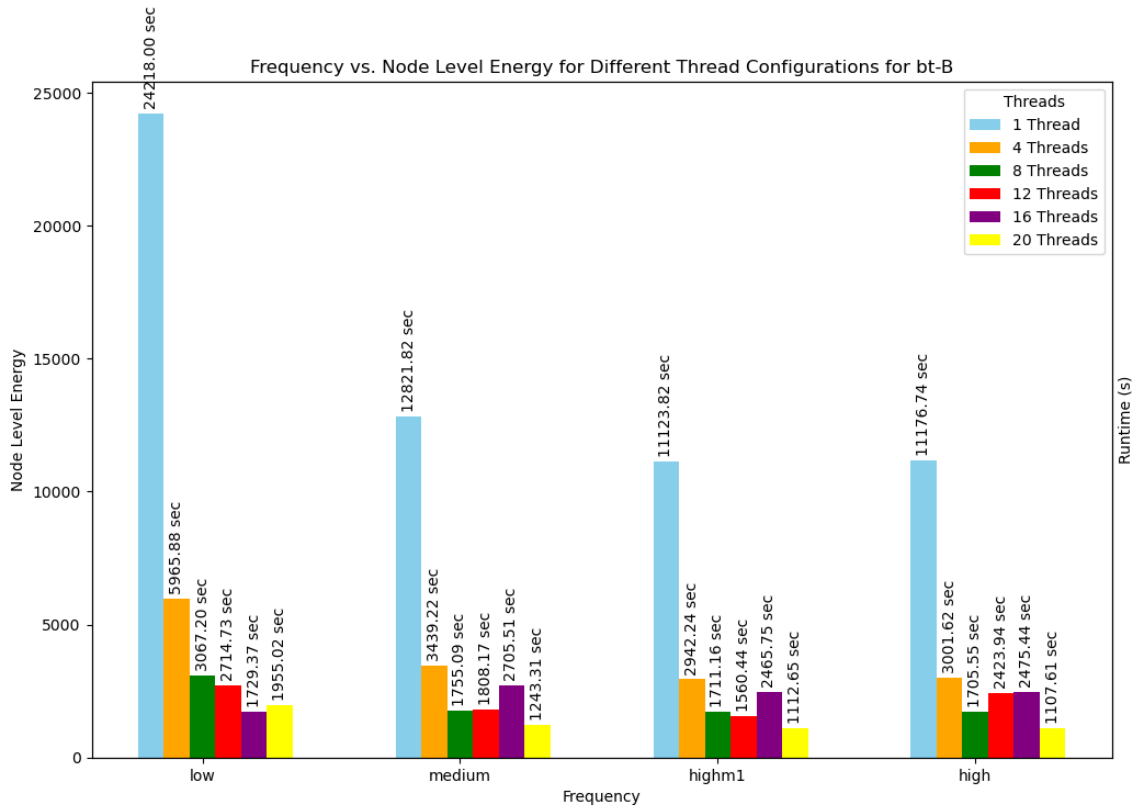


Figure 8.43: BT-B:Affect of CPU frequencies on the node level energy consumption for different thread count

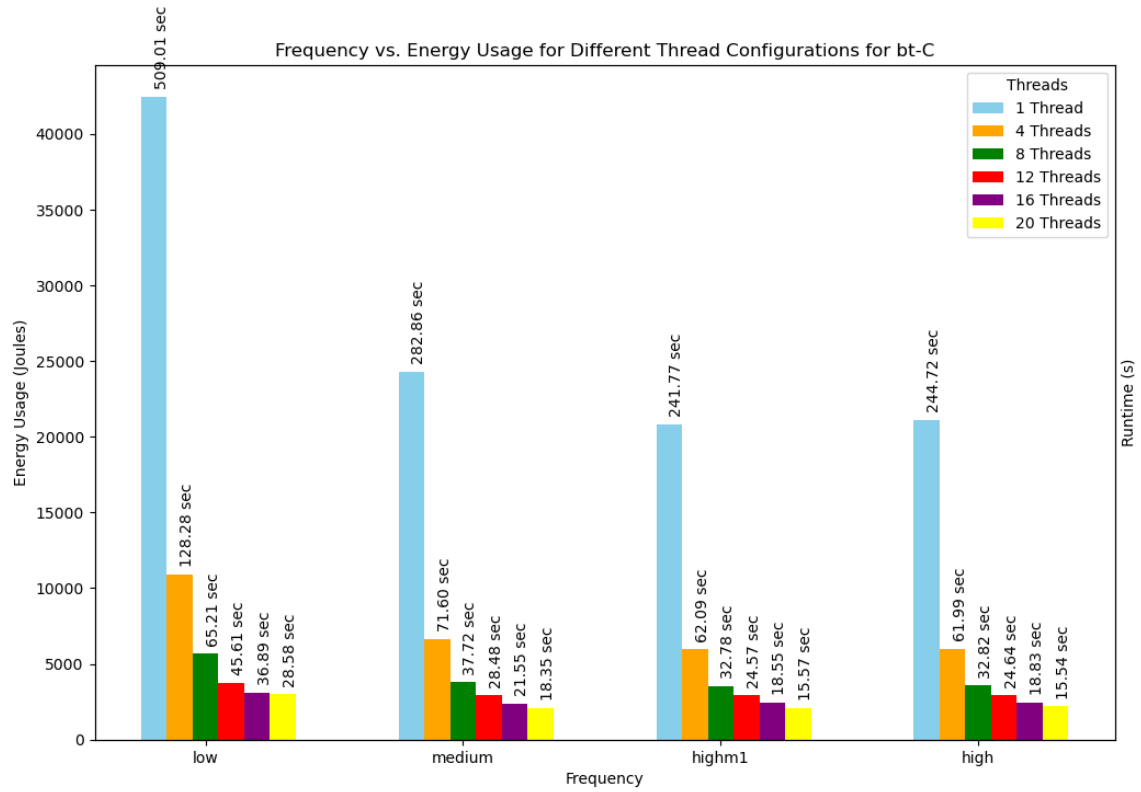


Figure 8.44: BT-C: Affect of CPU frequencies on the application level energy consumption for different thread count

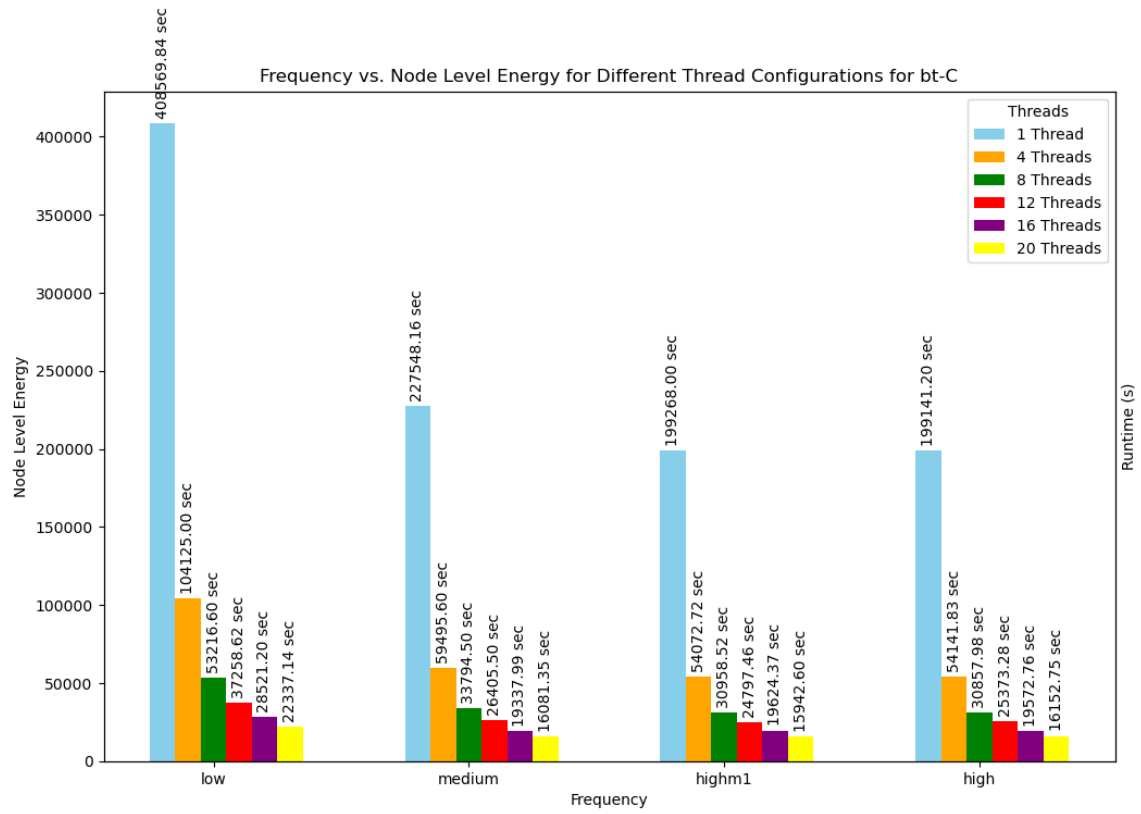


Figure 8.45: BT-C:Affect of CPU frequencies on the node level energy consumption for different thread count

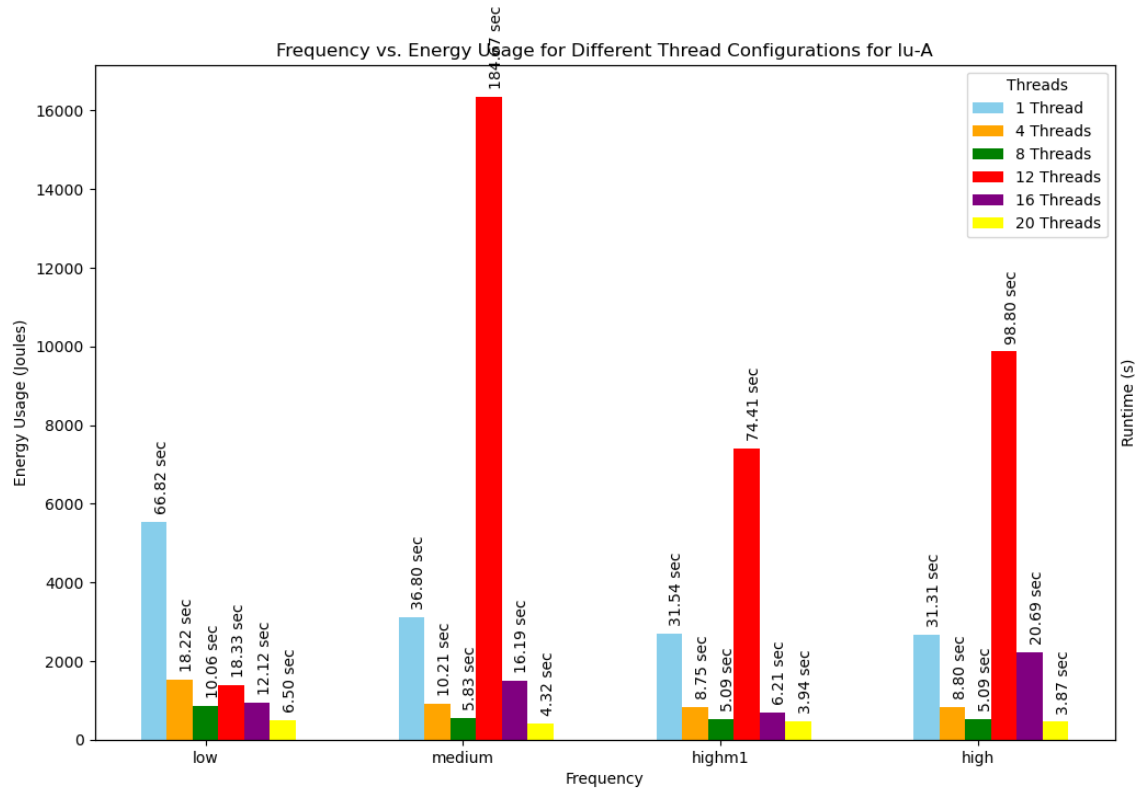


Figure 8.46: LU-A: Affect of CPU frequencies on the application level energy consumption for different thread count

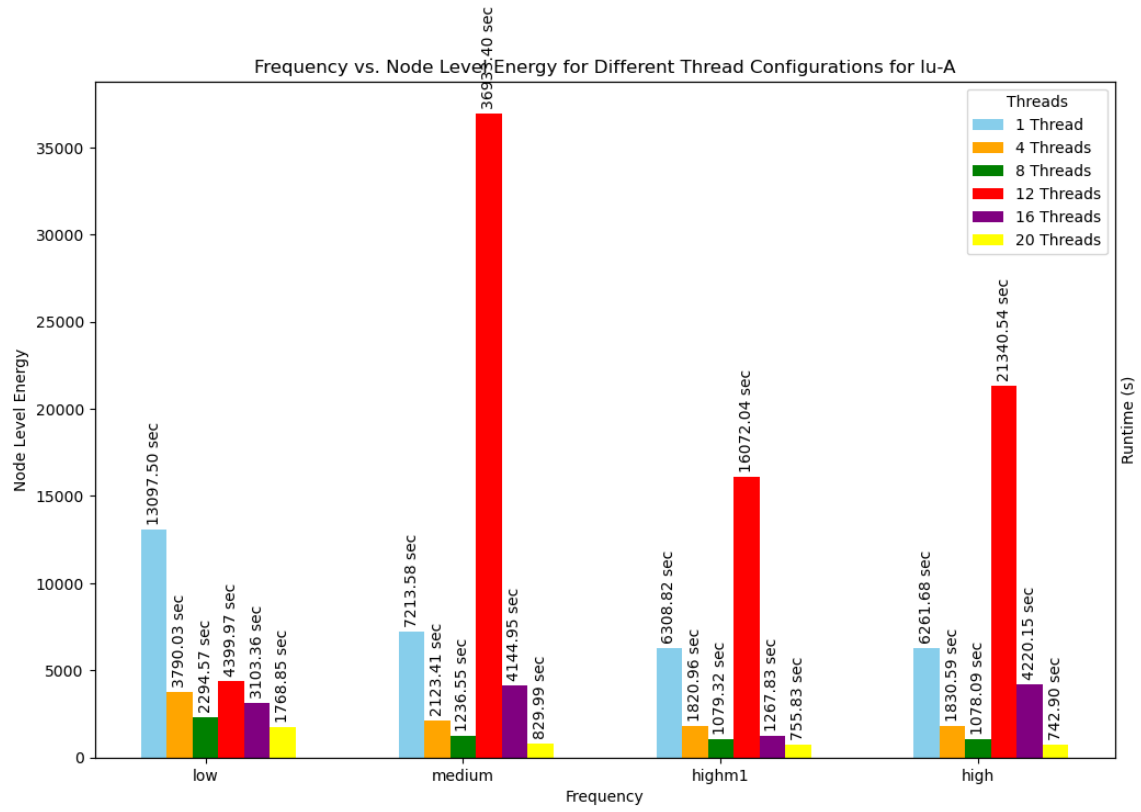


Figure 8.47: LU-A: Affect of CPU frequencies on the node level energy consumption for different thread count

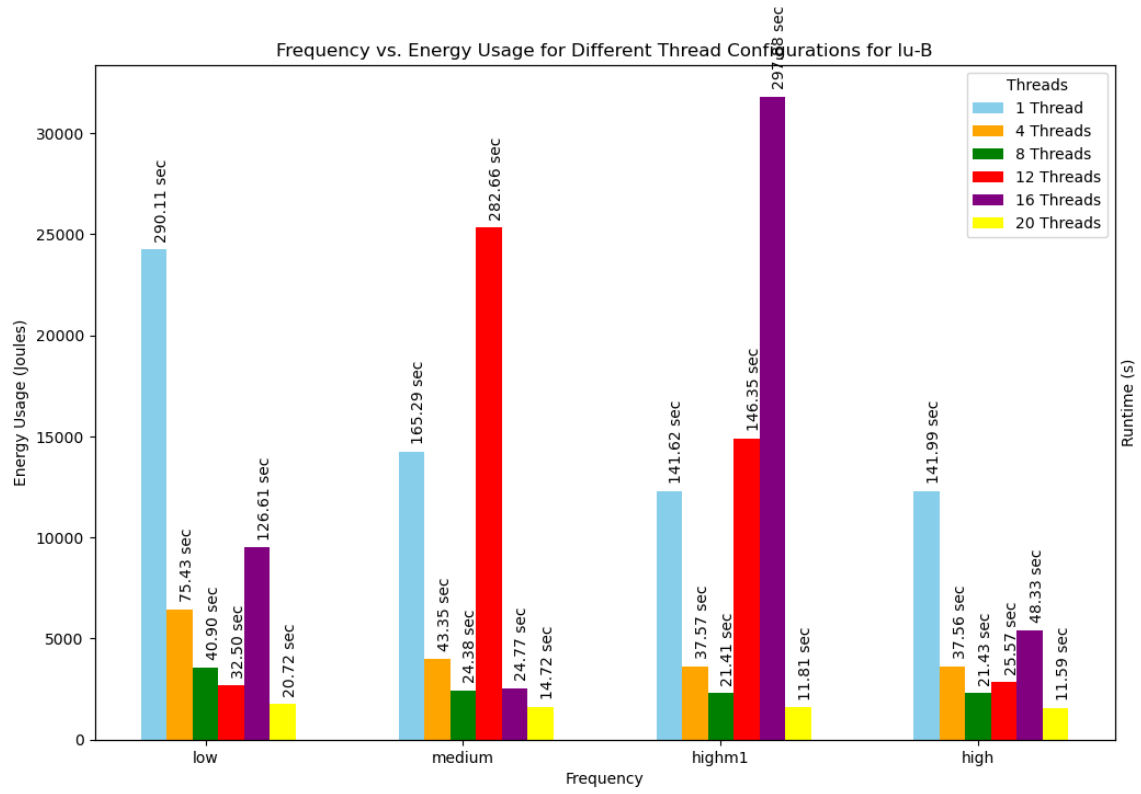


Figure 8.48: LU-B: Affect of CPU frequencies on the application level energy consumption for different thread count

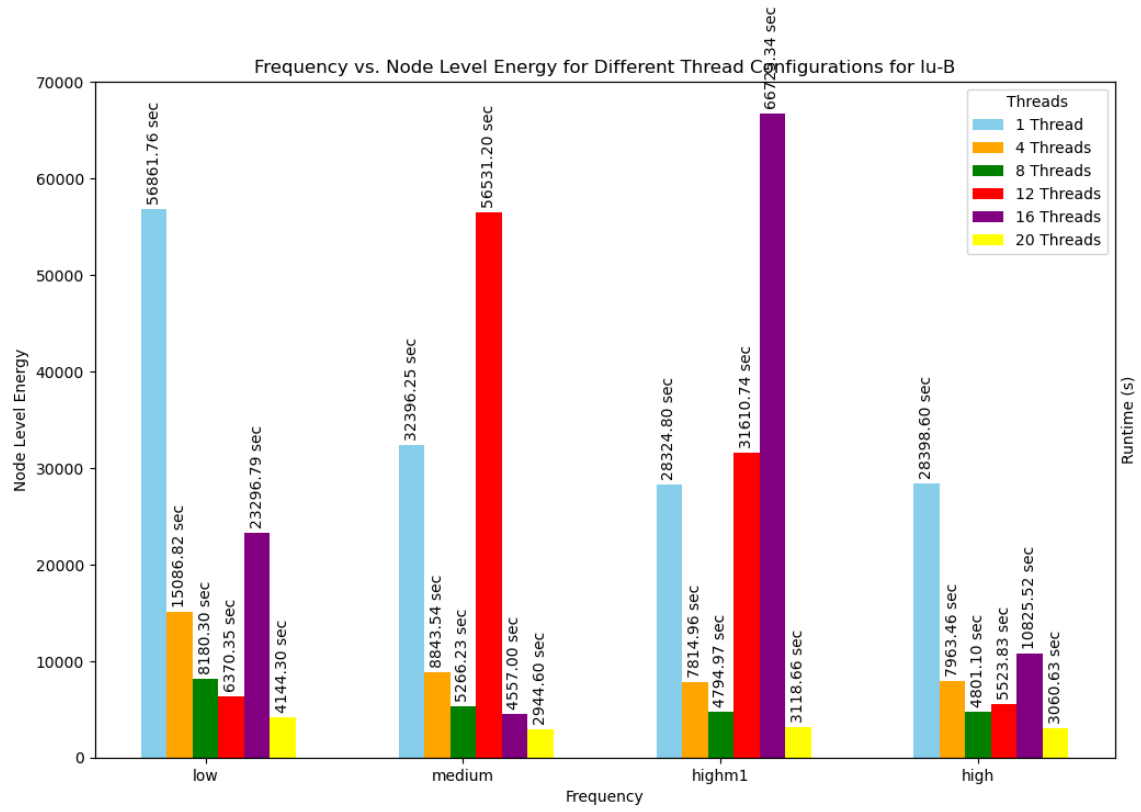


Figure 8.49: LU-B: Affect of CPU frequencies on the node level energy consumption for different thread count

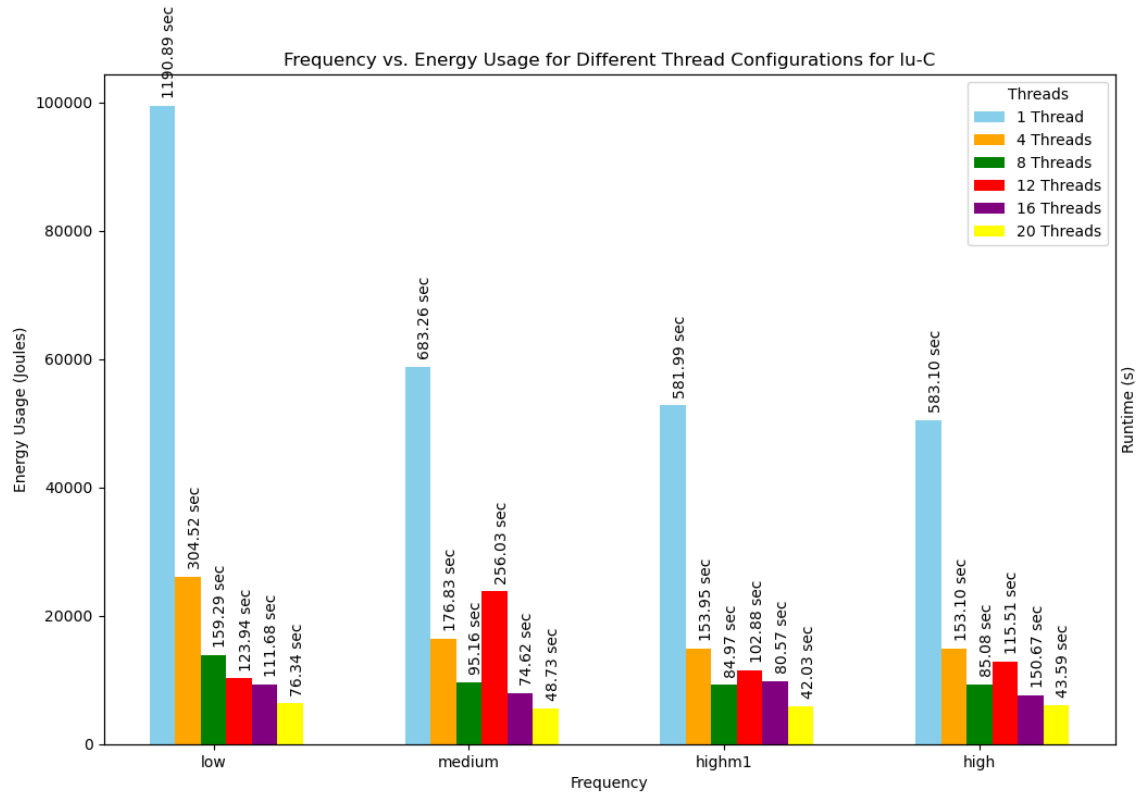


Figure 8.50: LU-C: Affect of CPU frequencies on the application level energy consumption for different thread count

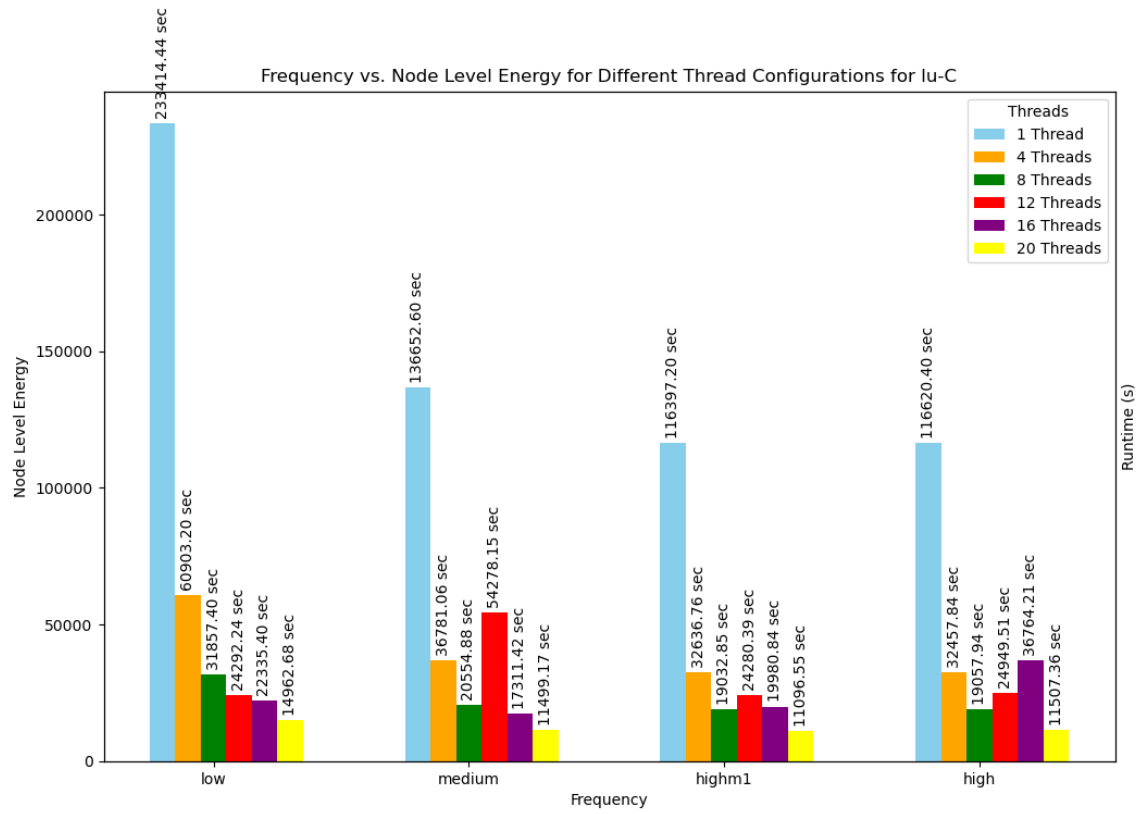


Figure 8.51: LU-C: Affect of CPU frequencies on the node level energy consumption for different thread count

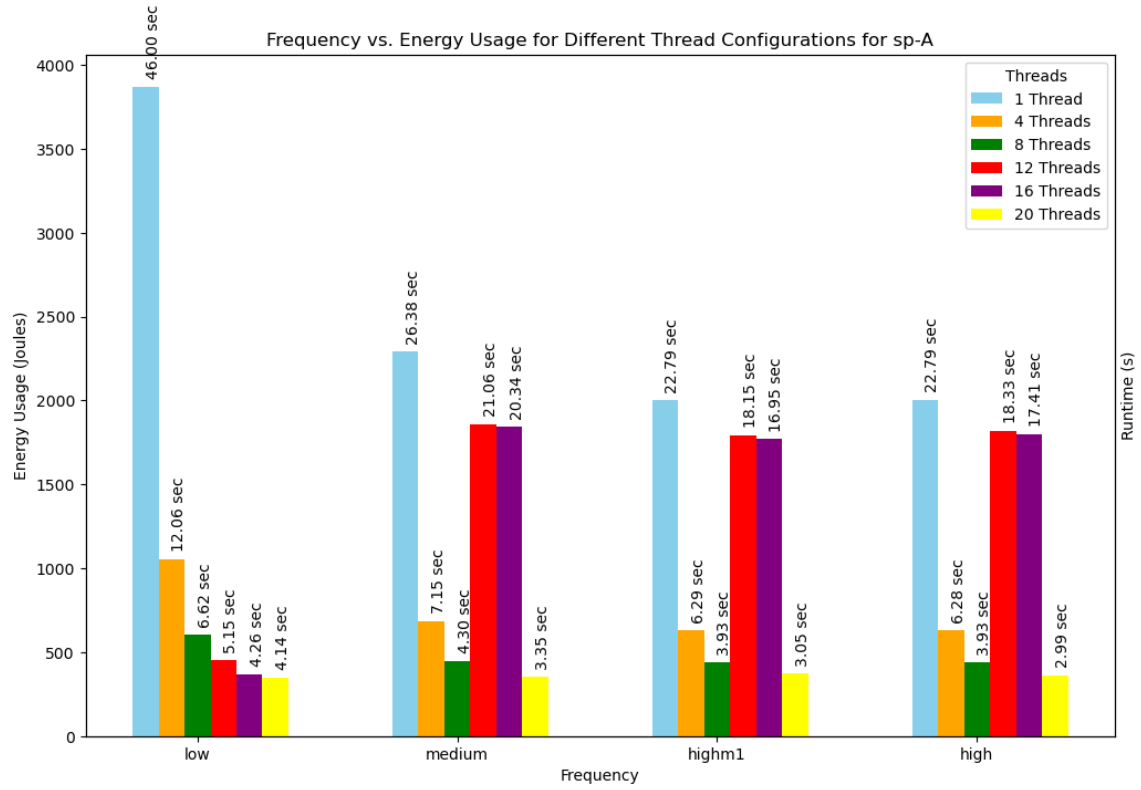


Figure 8.52: SP-A: Affect of CPU frequencies on the application level energy consumption for different thread count

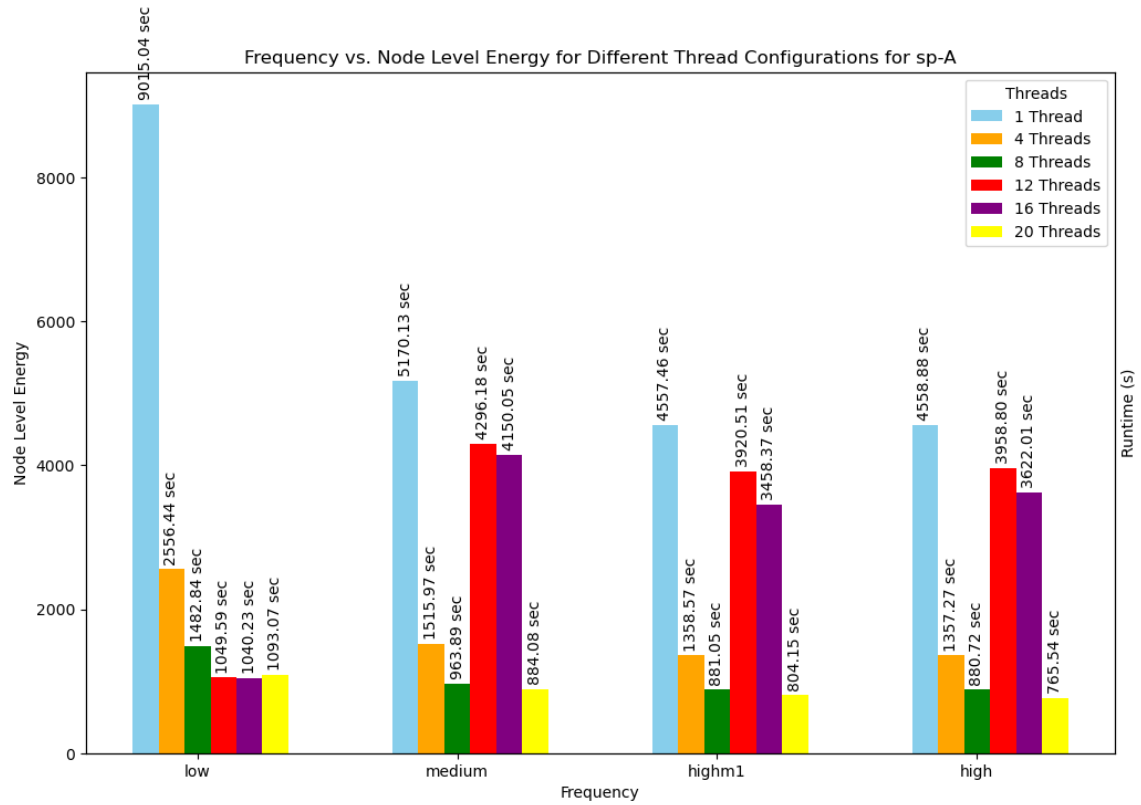


Figure 8.53: SP-A: Affect of CPU frequencies on the node level energy consumption for different thread count

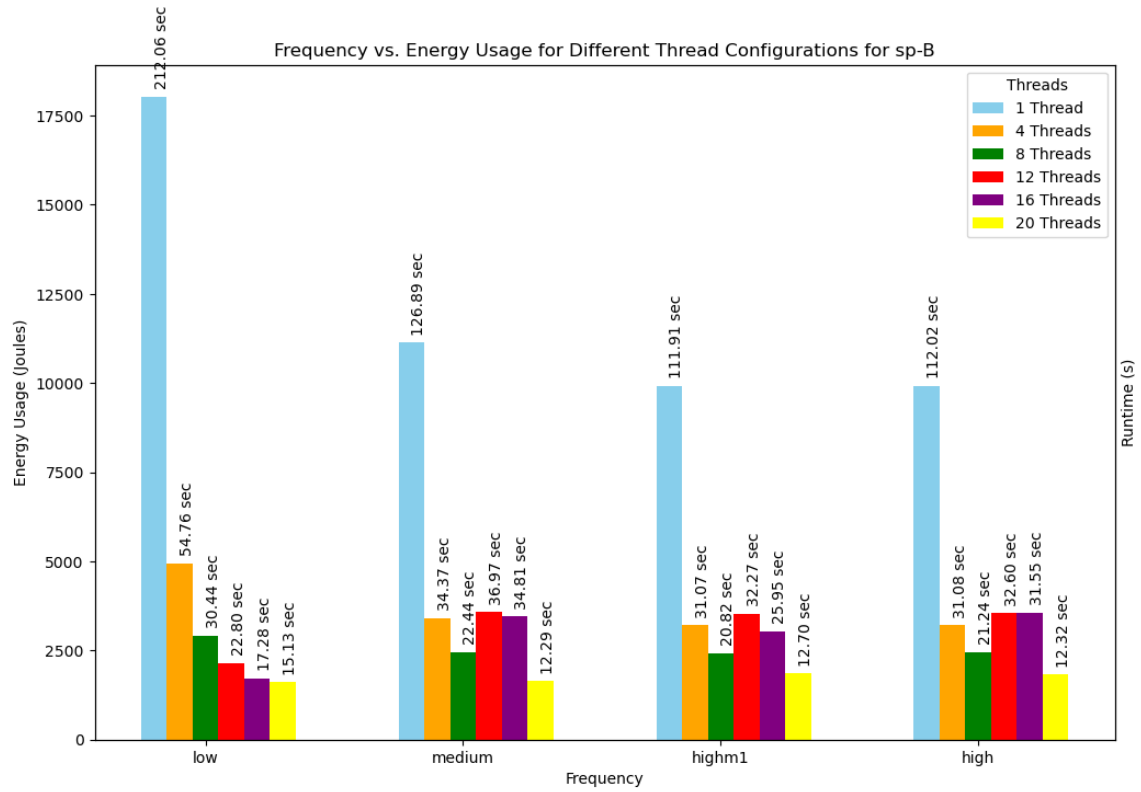


Figure 8.54: SP-B:Effect of CPU frequencies on the application level energy consumption for different thread count

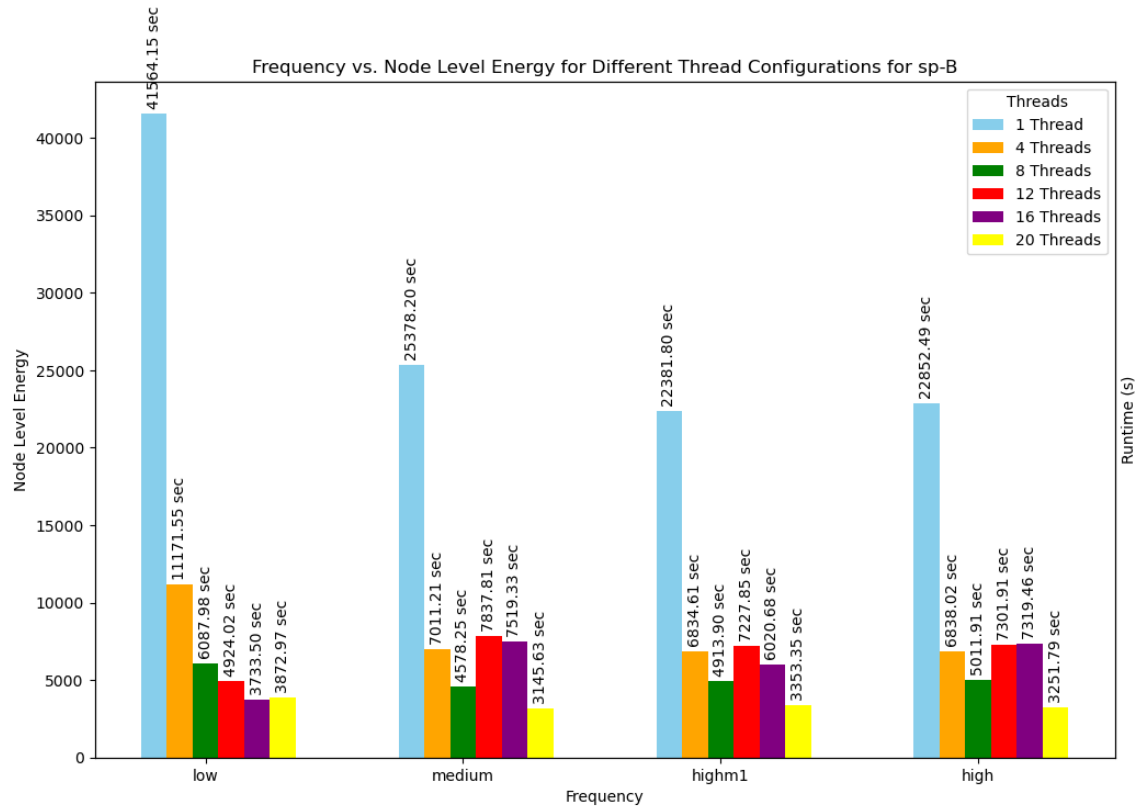


Figure 8.55: SP-B:Effect of CPU frequencies on the node level energy consumption for different thread count

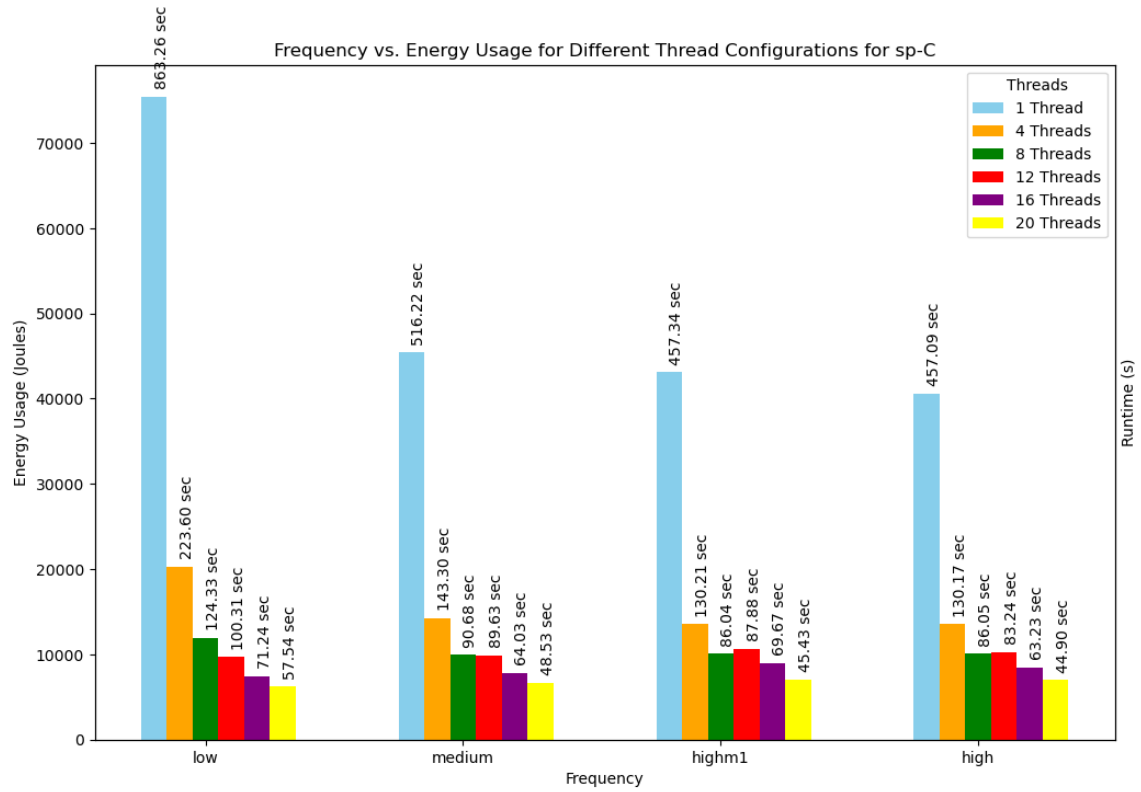


Figure 8.56: SP-C:Effect of CPU frequencies on the application level energy consumption for different thread count

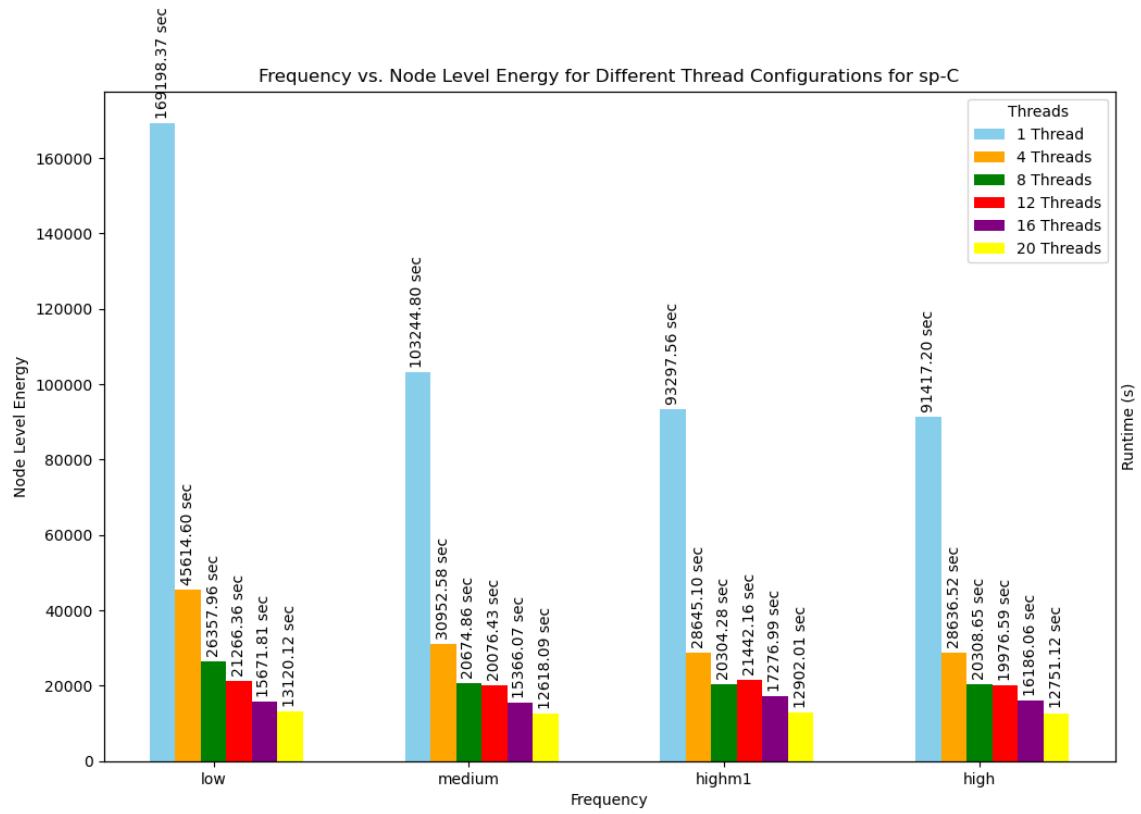


Figure 8.57: SP-C: Affect of CPU frequencies on the node level energy consumption for different thread count

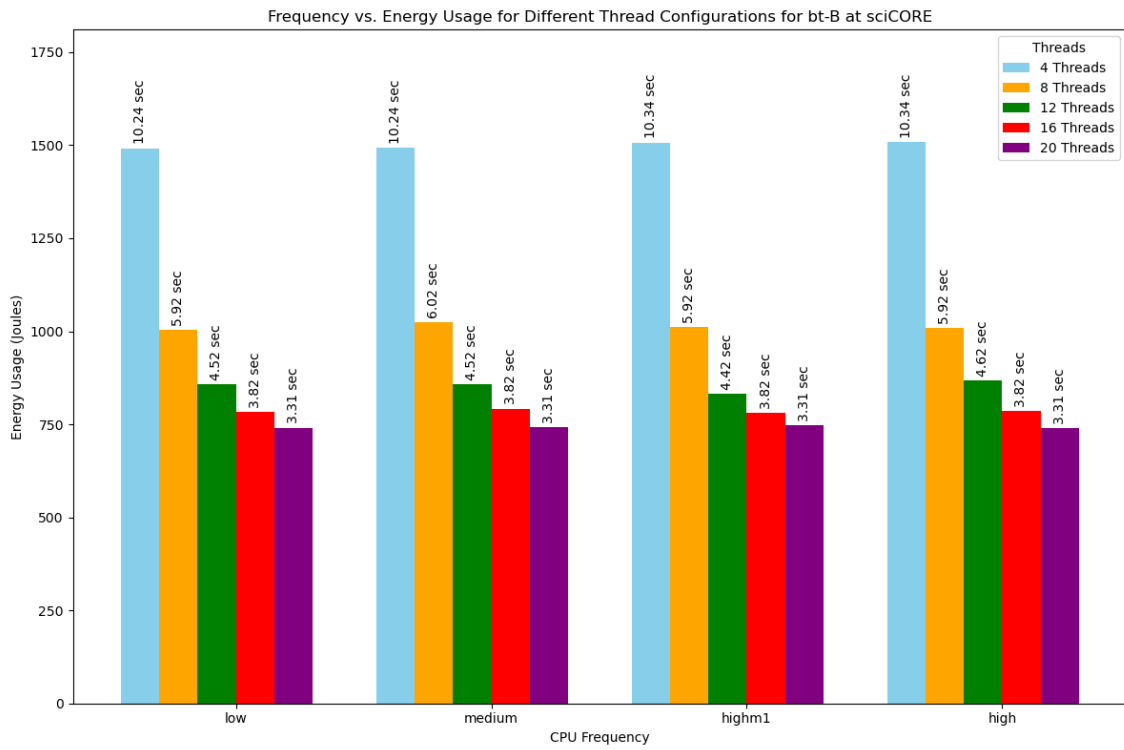


Figure 8.58: BT-A: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

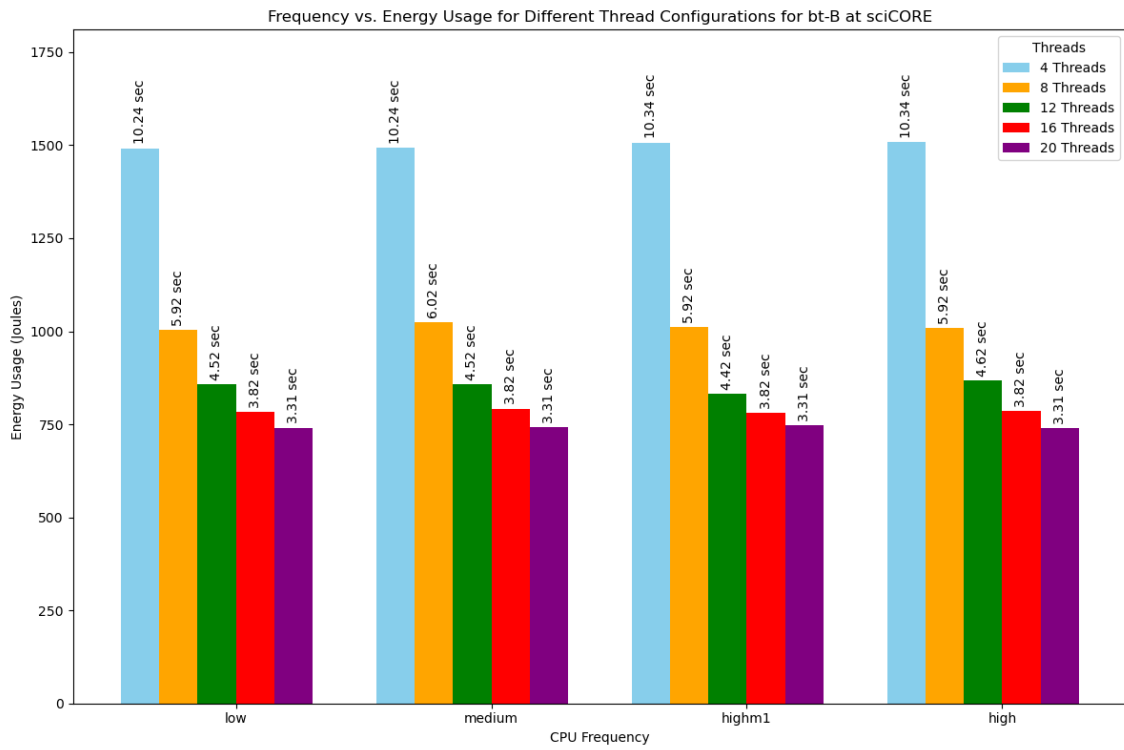


Figure 8.59: BT-B: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

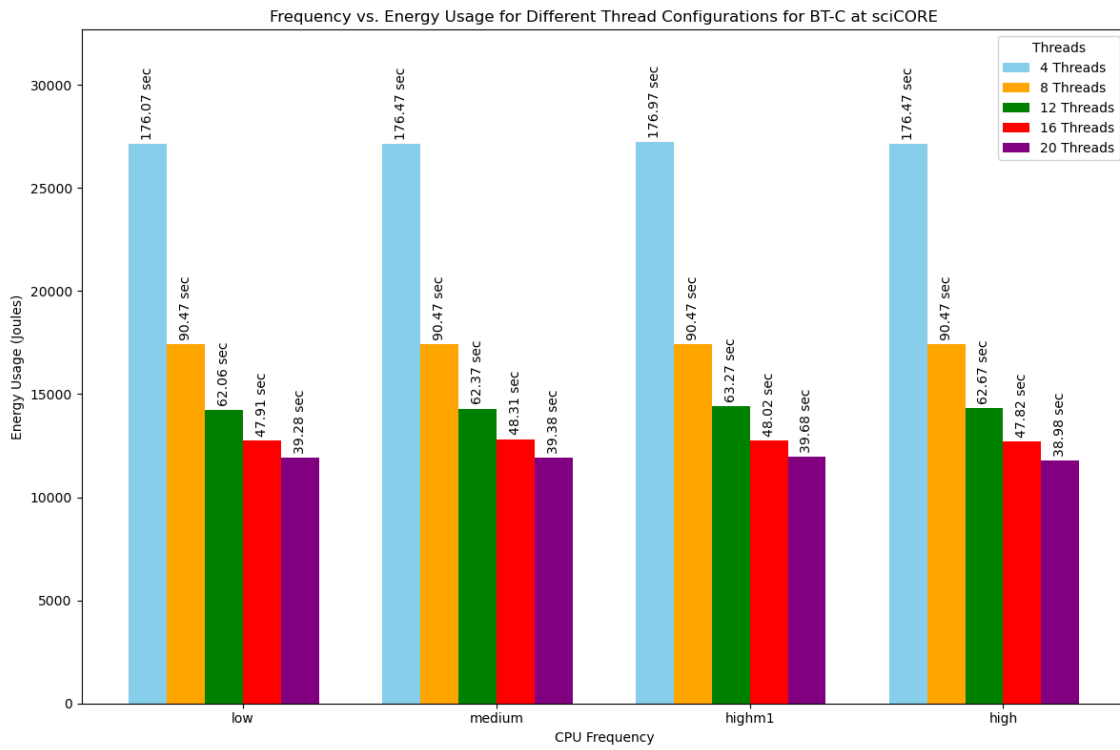


Figure 8.60: BT-C: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

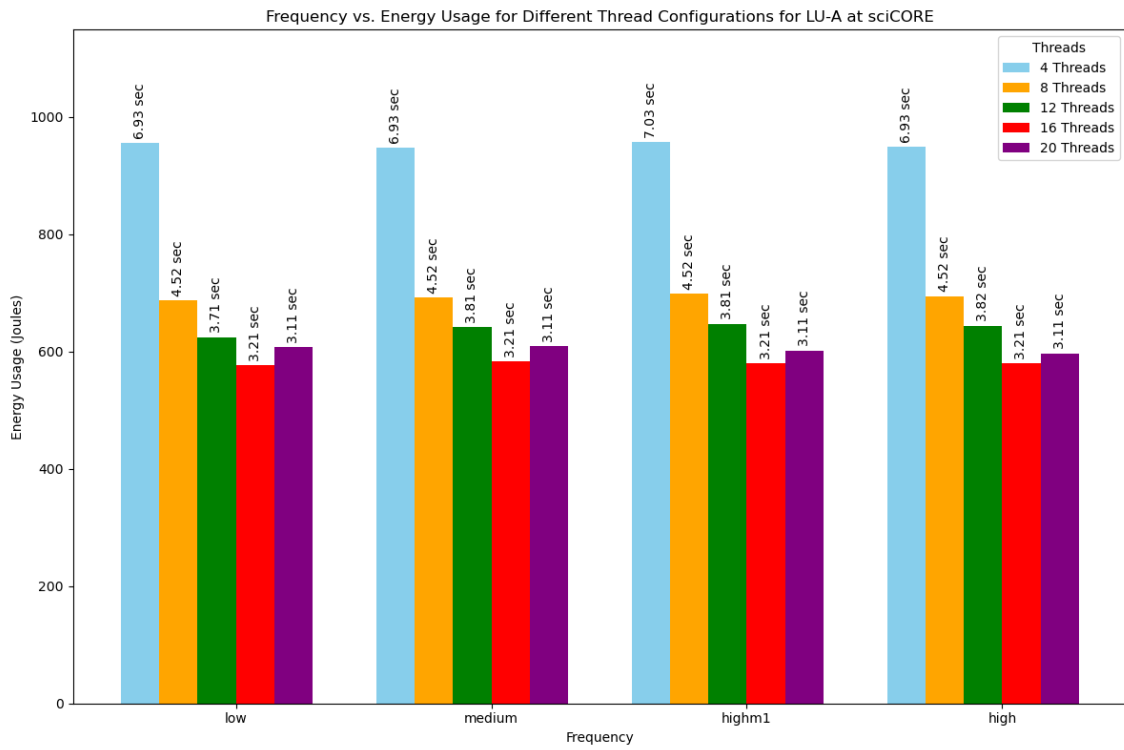


Figure 8.61: LU-A: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

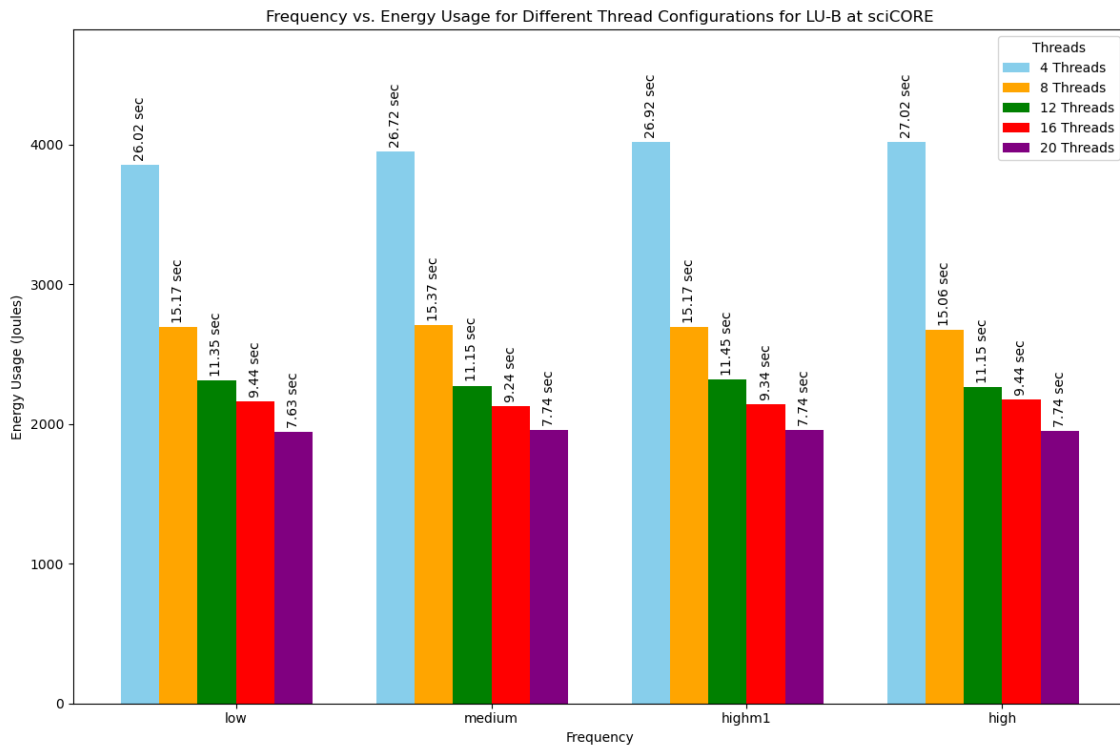


Figure 8.62: LU-B: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

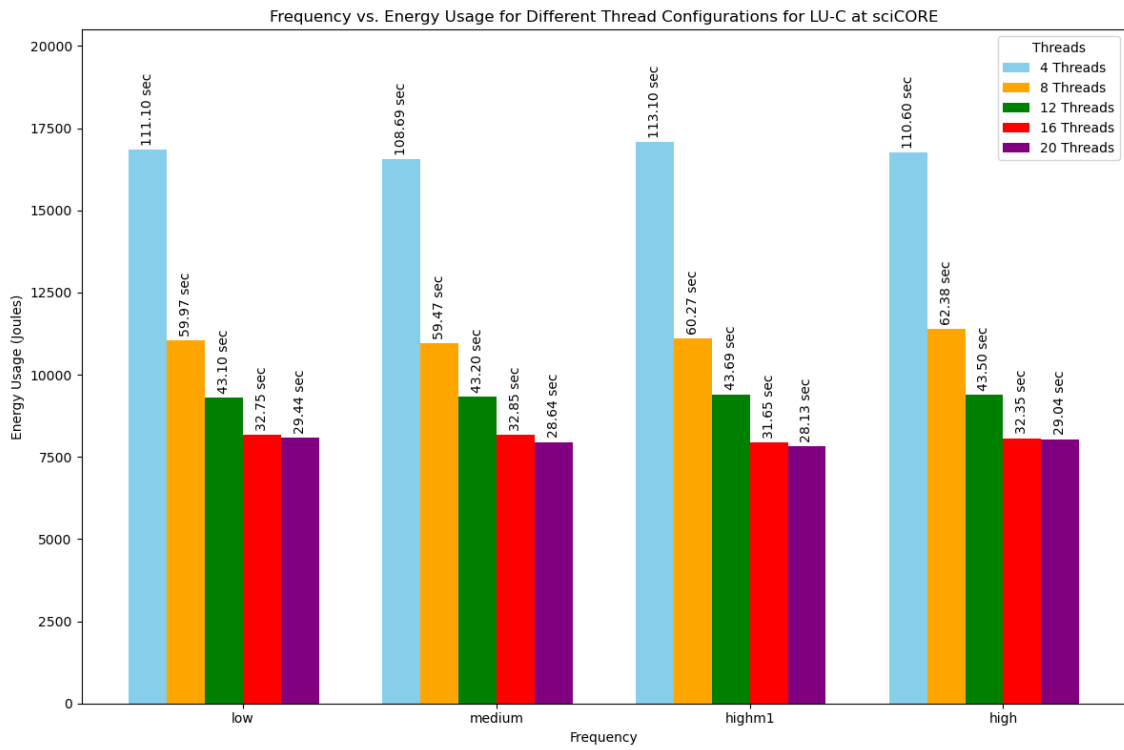


Figure 8.63: LU-C: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

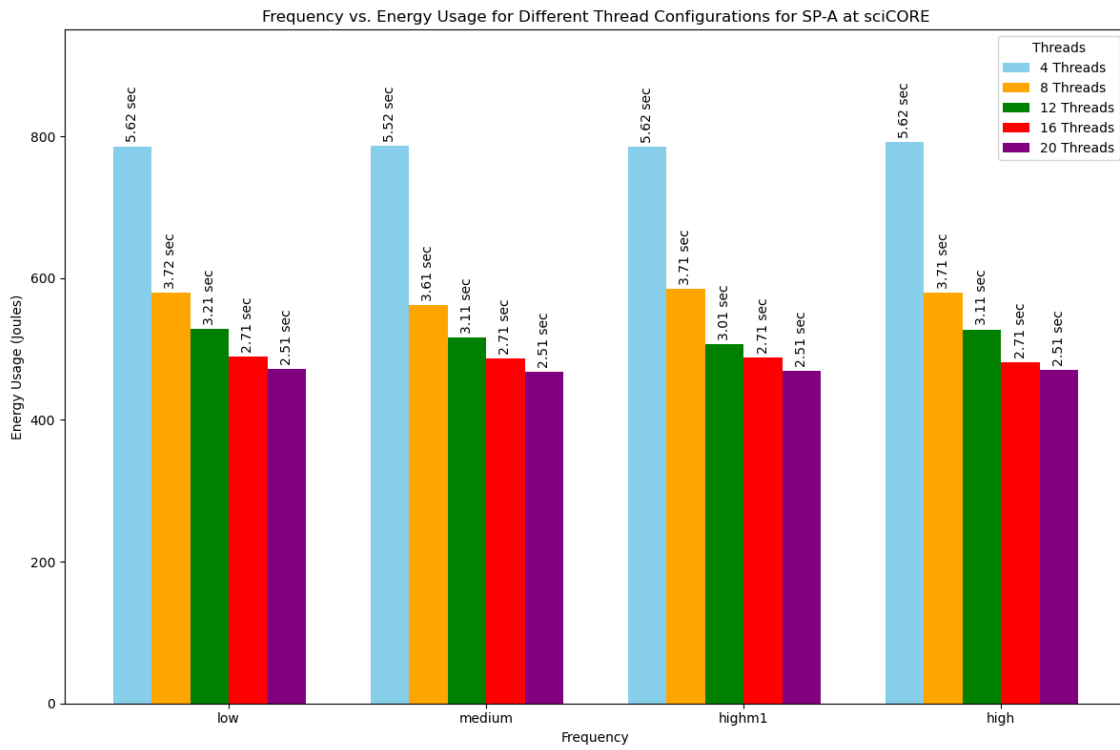


Figure 8.64: SP-A: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

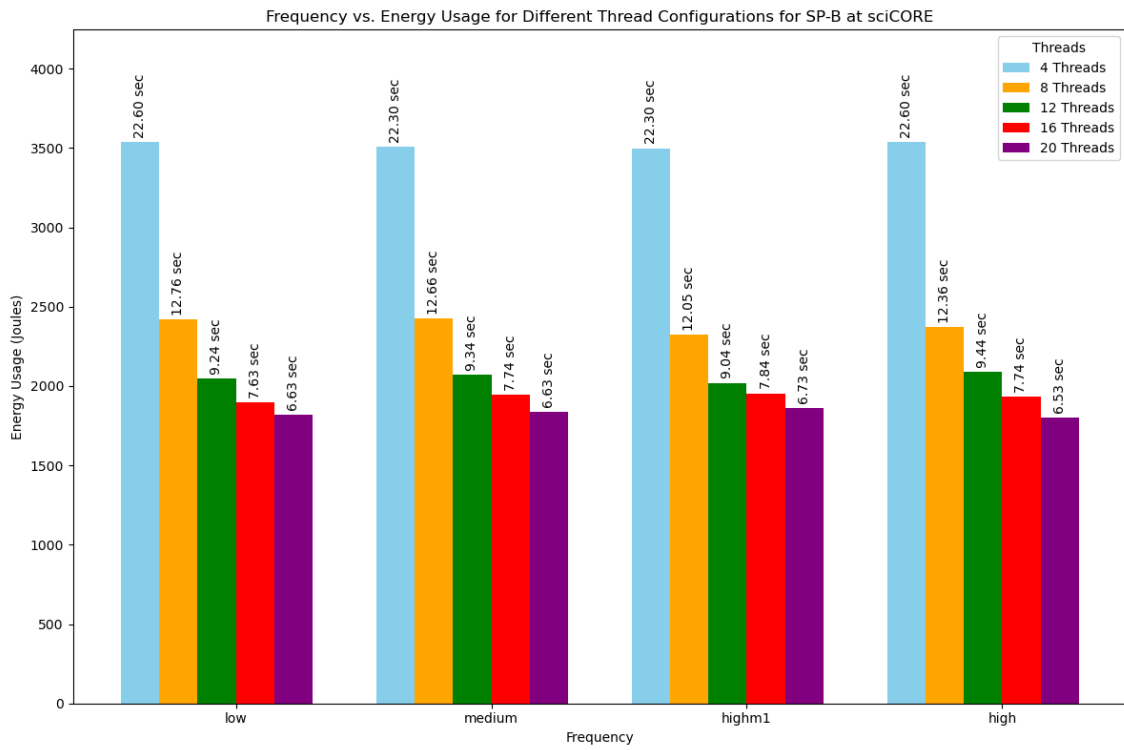


Figure 8.65: SP-B: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE

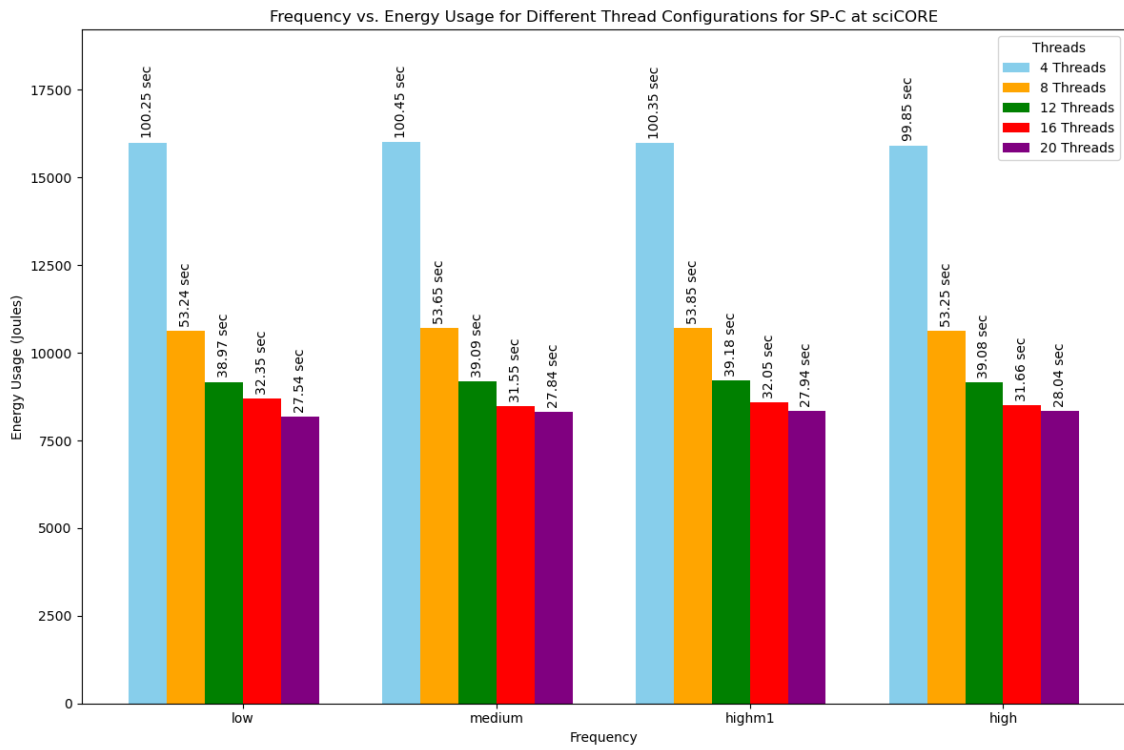


Figure 8.66: SP-C: Affect of CPU frequencies on the application level energy consumption for different thread count at sciCORE