Energy Consumption Analysis of CPU Tasks in Operating Systems

Nicholas Fantino-Dyer

Santa Clara University

CSEN 283

12/08/2024

Table of Contents

Abstract

Accurately measuring CPU performance and energy consumption is essential for optimizing system efficiency and identifying potential inefficiencies. This study investigates the inaccuracies present in widely-used monitoring tools, such as Intel Performance Counter Monitor (PCM) and Linux temperature sensors, across Windows and Linux environments. By collecting and analyzing CPU performance metrics under varying workloads and thread configurations on two CPU architectures—the Intel i7-7700K and i7-13700K—this research identifies significant discrepancies, including

negative energy values, CPU utilization exceeding 100%, and mismatched temperature readings. The findings indicate that Linux exhibits superior energy efficiency, with lower Energy per Instruction (EPI) and fewer Cache Misses per Joule compared to Windows. Despite implementing mitigation strategies such as extended sampling durations, data interpolation, and statistical filtering, numerous anomalies remained, revealing tool-specific limitations. Additionally, abnormal cases such as unexpected drops in energy consumption at high CPU loads and temperature plateaus were identified, indicating reliability issues in the monitoring tools. These results suggest the need for developing improved monitoring methodologies and validating data with additional measurement tools. This work outlines the limitations of existing CPU monitoring systems and their implications for performance analysis.

Introduction

Objective

As digital infrastructures grow, the energy consumption of computing systems has become an important issue, affecting both the environment and operational costs. CPUs, being at the core of computing devices, significantly contribute to the energy use of applications and operating systems. In my project, I will examine how CPUs consume energy during various operating system tasks, aiming to find patterns and methods to improve CPU energy efficiency.

Problem Statement

With the increasing demand for computational power, making operations more energy efficient is more important than ever. Current methods for predicting CPU power usage often rely only on utilization estimates, which can be inaccurate across different workloads, CPU architectures, and task complexities. These models usually assume a direct, linear relationship between CPU utilization and power consumption, which doesn't capture the complex and dynamic behavior of modern computing environments.

Several factors affect power consumption:

- **Task Complexity:** Different computational tasks use varying amounts of energy depending on their nature, such as input/output bound versus CPU bound processes.
- **CPU Architecture:** Variations in CPU designs, including differences among Intel, AMD, and ARM processors, influence how power is consumed.
- **Core Count and Types:** The number of active cores and the mix of performance and efficiency cores affect overall energy usage.
- **Power Management Settings:** Features that adjust CPU operating parameters to save energy add layers of complexity to power consumption patterns.

Moreover, energy efficiency can vary significantly between single core and multi core operations, influenced by factors like temperature, context switching overhead, core characteristics, and operating system behavior.

Importance in Operating Systems

Operating systems play a key role in managing CPU resources, handling many tasks with varying energy demands. By improving energy estimation and understanding consumption patterns, I can make better decisions in OS design and task scheduling to enhance energy efficiency. Accurate energy prediction is essential for optimizing OS performance, extending battery life, and reducing energy costs in data centers.

Why This Project Relates to Operating Systems

My project is closely connected to fundamental aspects of operating systems, such as process scheduling, resource management, and power management. I will analyze the CPU's energy consumption during OS level tasks, and I aim to find ways to develop more energy efficient operating systems.

Why Other Approaches Are Not Adequate

Existing methods for estimating CPU energy consumption have several shortcomings:

1. **Dependence on CPU Utilization Alone:** Many models predict power usage based only on CPU utilization percentages, assuming a direct linear relationship with energy consumption. This simplification often leads to inaccuracies because it doesn't account for different task complexities and CPU architectures.
2. **Neglect of Dynamic Factors:** Simple models often overlook factors like temperature changes and workload fluctuations. These elements can significantly affect energy consumption but are not considered, reducing the models' predictive accuracy.

3. **Inadequacy for Multi Core and Hybrid Architectures:** Existing models may not accurately predict energy usage in multi core or hybrid CPU architectures. Interactions between cores, context switching, and complex task scheduling greatly influence power consumption in these environments.
4. **Tool Reliability Issues:** During preliminary analyses, I discovered that widely-used monitoring tools like Intel's Performance Counter Monitor (PCM) exhibited significant inaccuracies. PCM occasionally reported negative energy values and CPU utilizations exceeding 100%, undermining its reliability for precise energy measurements. Additionally, some temperature sensors provided inconsistent or imprecise data, further complicating accurate energy consumption assessments.

Why I Think my Approach Is Better

To address these shortcomings, I employed the following strategies:

- **Integration of Dynamic Factors:** My analysis incorporated temperature variations and task complexity, enhancing the precision of energy consumption assessments.
- **Architecture-Neutral Design:** I designed the analysis framework to be compatible with various CPU architectures, including Intel, AMD, and ARM processors, ensuring broad applicability.
- **Practical Implementation:** By utilizing existing software tools and methodologies without requiring specialized hardware or elevated system privileges, I made the analysis accessible and straightforward to adopt.

- **Critical Evaluation of Monitoring Tools:** Given the identified reliability issues with tools like PCM and certain sensors, I conducted a thorough evaluation of these tools' accuracy. This involved cross-validating energy measurements using alternative methods and identifying the extent of their inaccuracies, which informed the robustness of my subsequent analysis.

Proposed Solution

This study proposed a comprehensive approach to estimating CPU energy consumption by addressing the deficiencies of existing methodologies. The key components of the solution encompassed:

- **Benchmark Development and Task Simulation:** Although I initially intended to create benchmarks that emulated typical operating system-level tasks, I instead utilized existing benchmarks to assess energy consumption under diverse task scenarios and workloads.
- **Evaluation of Estimation Formulas:** I tested a variety of CPU energy estimation formulas, including those that accounted for dynamic influences such as temperature and workload variations. The accuracy of these formulas was evaluated across different CPU architectures and workload types to identify the most reliable models.
- **Comparative Analysis Across CPU Architectures and Generations:** By analyzing energy consumption patterns across various CPU models, including both ARM and x86 architectures, I identified trends and optimal conditions for

energy efficiency. This analysis informed hardware selection and system design considerations.

- **Assessing Operating System Impact on Energy Consumption:** I conducted analyses on multiple operating systems to understand how operating system-level power management and scheduling policies affected CPU energy usage. This revealed how different operating system designs influenced energy efficiency.
- **Exploring Hybrid CPUs and Idle Power Management:** I investigated systems equipped with hybrid CPUs that combined performance and efficiency cores. The study analyzed how task allocation to different core types impacted overall energy consumption, particularly during periods of low demand.
- **Critical Assessment of Monitoring Tools:** Recognizing the limitations of tools like PCM, I evaluated their performance and reliability in capturing accurate energy consumption data. This assessment highlighted the need for more dependable measurement techniques and informed the interpretation of the energy trends observed.

Through these initiatives, the objective was to develop an estimation model capable of providing accurate, real-time predictions of CPU energy consumption. This model aimed to support energy-aware optimizations in operating systems, contributing to more sustainable and cost-effective computing environments.

Theoretical Bases and Literature Review

Related Research

Analysis of RAPL Measurement Techniques

In their 2024 study, Raffin and Trystram explored various software methods for accessing RAPL (Running Average Power Limit) energy measurements. They compared several approaches:

- **Model Specific Registers (MSR):** This method involves direct, low level access to energy counters.
- **Power Capping Framework (powercap):** Provided higher level access through the sysfs virtual file system.
- **Performance Counters Subsystem (perf events):** Used event oriented performance monitoring with safeguards against counter overflows.
- **Extended Berkeley Packet Filter (eBPF):** A mechanism that collects data from within the kernel.

They identified several challenges when using RAPL for energy measurement:

- **Technical Complexity:** Methods like MSR require deep technical knowledge and elevated system privileges.
- **Performance Impact:** Different approaches can affect system performance in various ways.
- **Measurement Accuracy:** Problems such as counter overflows and timing issues can make measurements less reliable.

Strengths of Their Research:

- **Comprehensive Evaluation:** They provided a detailed comparison of different RAPL access methods.
- **Practical Advice:** Their work offered guidance on selecting appropriate tools based on specific needs.

Limitations:

- **Hardware Specificity:** The study focused on CPUs that support RAPL, mainly Intel and some AMD processors, and doesn't include architectures like ARM.
- **Linux Dependency:** The methods rely on Linux kernel interfaces, which limits their applicability to other operating systems.

Energy Measurement Frameworks for Android

In 2020, Myasnikov and colleagues reviewed various energy consumption measurement frameworks for Android applications. They categorized existing tools into two main methods:

- **Direct Measurement:**
    - **External Meters:** Used devices like multimeters to measure voltage and current directly from the device's battery.
    - **Internal Meters:** Accessed built in power sensors through Android APIs.
- **Indirect Measurement (Model Based):**
    - **Working Time Model:** Estimated energy use based on how long hardware components are active.

    - **Instruction Energy Model:** Calculated energy based on the number and types of instructions executed.
    - **Method/API Call Energy Model:** Estimated energy consumption by analyzing how the application uses system or API calls.

They found several issues:

- **Inconsistency:** Different methodologies and units of measurement make it hard to compare the accuracy of frameworks.
- **Limited Access:** Few frameworks are open source, which restricts accessibility.
- **Lack of Detailed Methodologies:** Many studies do not provide detailed methodologies, making reproducibility difficult.

Strengths of Their Research:

- **Detailed Classification:** Provided developers with a clear understanding of energy measurement tools available for Android.
- **Highlighting Challenges:** Emphasized the need for standard methodologies and accessible tools.

Limitations:

- **Mobile Focus:** The research mainly addressed mobile applications, which might not apply directly to desktop or server CPUs.
- **Measurement Constraints:** Like RAPL based tools, some methods require specific hardware or elevated privileges.

Energy Efficient Scheduling in Heterogeneous Systems

In 2020, Tang and Fu developed a model for energy aware scheduling in systems that use both CPUs and GPUs. They focused on:

- **Minimizing Energy Use:** By efficiently distributing jobs between CPUs and GPUs.
- **Balancing Workloads:** Allocated computational tasks to optimize energy consumption.

While their research is specific to systems that combine CPUs and GPUs, it provided useful insights for my work, such as:

- **Understanding Task Scheduling:** Learning how different scheduling methods impact energy use.
- **Applying Techniques to Multi Core CPUs:** Adapting their strategies to improve energy efficiency in CPU only systems.

Advantages of Their Work:

- **Optimization Methods:** Offers ways to schedule tasks more efficiently to save energy.
- **Relevance to High Performance Systems:** Provided insights that are important for environments where energy efficiency is a key concern.

Limitations:

- **Specific Focus:** Their model is tailored to heterogeneous systems and may not directly apply to systems with only CPUs.
- **Assumptions of Controlled Settings:** Their work might not consider the variability found in general-purpose computing systems.

My Solution to Solve This Problem

I aim to develop a CPU energy consumption estimation model that overcomes the limitations identified in existing approaches, providing a practical and accurate tool for predicting energy consumption based on utilization and other dynamic factors.

Architecture Agnostic Estimation Model

- **Universal Applicability:** Designed to work across various CPU architectures, including Intel, AMD, and ARM processors.
- **No Special Hardware Requirements:** Eliminates dependency on hardware specific features like RAPL, making it suitable for a wider range of devices.
- **Temperature Fluctuations:** Considers thermal effects that influence CPU performance and energy usage.
- **Core Types and Task Complexity:** Differentiates between performance and efficiency cores and accounts for the nature of computational tasks.

Estimation Over Direct Measurement

- **Software Based Estimation:** Relies on predictive models rather than direct hardware measurements, making it more accessible and less intrusive.

- **Real Time Estimation:** Enables immediate feedback during application development and execution, facilitating energy aware optimizations.

Standardized Experimental Methodology

- **Detailed Methodological Framework:** Provides comprehensive guidelines for experimental setup, data collection, and analysis.
- **Accuracy Assessment Protocols:** Implements standardized procedures for evaluating and reporting the accuracy of energy estimations.

Where my Solution Differs from Others

- **Comprehensive Integration:** Combines multiple influencing factors into a single estimation model, offering a more holistic approach than models focusing solely on utilization or specific hardware features.
- **Accessibility and Practicality:** Designed to be easily adopted by developers and system administrators without requiring specialized hardware or elevated privileges.
- **Standardization Efforts:** Aims to establish standardized measurement units and methodologies to facilitate comparison, replication, and broader adoption.

Why my Solution Is Better

- **Broader Applicability:** Suitable for a wide range of systems, including those without RAPL support, enhancing its utility across different computing environments.

- **Enhanced Accuracy:** By integrating dynamic system factors, my model provides more precise energy consumption estimates.
- **Ease of Use:** Simplifies the process of energy estimation, making it accessible to practitioners without specialized expertise.
- **Contribution to Energy Efficiency:** Enables more effective energy aware scheduling and optimization, leading to reduced energy consumption and environmental impact.

Hypothesis

My primary hypothesis posited that multi-core processing is more energy efficient than single-core processing under specific tasks. Building upon this, I formulated several additional hypotheses to explore various facets of CPU energy consumption:

Hypothesis on Task Complexity

Hypothesis: High-complexity tasks consumed more energy on single-core processing than on multi-core processing due to increased strain and higher operating temperatures.

Testing Approach: I ran benchmarks with varying complexity levels, comparing energy usage between single-core and multi-core setups to assess how task complexity influenced energy efficiency.

Hypothesis on Idle Power Consumption

Hypothesis: CPUs exhibited greater energy efficiency in multi-core configurations during low utilization tasks because additional cores could remain inactive, thereby reducing overall power draw.

Testing Approach: I compared energy consumption of idle or low-load tasks across single-core and multi-core settings to determine the impact of core inactivity on energy savings.

Hypothesis on Estimation Formula Sensitivity

Hypothesis: Estimation formulas that incorporated Dynamic Voltage and Frequency Scaling (DVFS) data provided more accurate predictions for energy consumption than those that did not, particularly under fluctuating workloads.

Testing Approach: I applied estimation formulas with and without DVFS data to benchmark results, evaluating their accuracy against actual measurements to identify the most reliable prediction models.

Hypothesis on Temperature Impact

Hypothesis: As CPU temperature increased, energy consumption also rose due to thermal inefficiencies, with single-core configurations being more susceptible to temperature-driven inefficiencies than multi-core configurations.

Testing Approach: I measured and compared energy usage at different CPU temperatures for single-core and multi-core operations to assess the relationship between temperature and energy consumption.

Hypothesis on Core Type Utilization

Hypothesis: Hybrid CPUs achieved lower energy consumption by routing tasks to efficiency cores during low to moderate demand tasks.

Testing Approach: I tested benchmarks on hybrid CPUs, analyzing energy consumption when tasks were assigned to different core types to evaluate the benefits of core specialization.

Hypothesis on Task Switching Overheads

Hypothesis: Frequent context switching in multi-threaded tasks led to higher energy consumption on multi-core setups, potentially negating some benefits of parallel processing.

Testing Approach: I ran tasks with varying levels of context switching, comparing energy usage on single-core versus multi-core setups to determine the impact of task switching on energy efficiency.

Hypothesis on Benchmark Repeatability

Hypothesis: Repeated executions of the same benchmark tasks showed consistent energy usage patterns on newer CPUs, while older CPUs exhibited greater variability due to wear and reduced thermal efficiency.

Testing Approach: I conducted repeated runs of identical tasks across different generations of CPUs, measuring the consistency of energy usage to evaluate the reliability and stability of energy consumption trends over time.

Hypothesis on CPU Architecture Differences

Hypothesis: ARM-based CPUs consumed less energy than x86 CPUs for the same tasks due to architectural efficiencies, particularly in low to moderate complexity workloads.

Testing Approach: I compared energy consumption across ARM and x86 CPUs using identical benchmarks to identify architectural influences on energy efficiency.

Hypothesis on Operating System Impact

Hypothesis: Energy consumption varied by operating system due to differences in task scheduling, power management, and idle state policies.

Testing Approach: I ran identical tasks on the same hardware across different operating systems, such as Windows and Linux, and compared energy usage to assess the influence of OS-level management on energy efficiency.

Hypothesis on Memory-Intensive Task Efficiency

Hypothesis: For memory-intensive tasks, CPUs with larger cache sizes demonstrated higher energy efficiency by reducing the need to access external memory.

Testing Approach: I executed memory-intensive benchmarks on CPUs with varying cache sizes, comparing energy consumption rates to determine the effect of cache capacity on energy efficiency.

Hypothesis on Monitoring Tool Reliability

Hypothesis: Commonly used monitoring tools like Intel's Performance Counter Monitor (PCM) and certain temperature sensors were unreliable, leading to inaccurate energy consumption data.

Testing Approach: I critically evaluated the reliability of PCM and various sensors by cross-validating their energy measurements against alternative methods. This involved identifying discrepancies such as negative energy values and utilization rates exceeding 100%, and assessing their impact on the accuracy of the observed energy trends.

Each hypothesis was tested by measuring energy consumption in benchmarked CPU tasks, comparing single-core and multi-core executions, and analyzing data across various CPU models. Additionally, I evaluated the reliability of monitoring tools to ensure the integrity of the energy consumption data used in my analysis.

Methodology

Data Collection

I utilized Intel's Running Average Power Limit (RAPL) through the Performance Counter Monitor (PCM) and HeavyLoad for Windows, alongside PCM, stress-ng, and temperature sensors for Linux to capture real-time energy data. Each CPU task was isolated within a controlled environment to ensure precise measurement of energy consumption. The tasks encompassed single-threaded and multi-threaded operations, memory-intensive processes, and varying levels of CPU load. Additionally, I assessed

the reliability of these monitoring tools, identifying inconsistencies such as negative energy values reported by PCM and inaccuracies in temperature sensor data.

Algorithm Design

Rather than developing a custom benchmark suite, I employed existing stress-testing tools to simulate typical operating system-level tasks. On Windows, HeavyLoad was used to generate CPU load, while on Linux, stress-ng was utilized to create varying workloads. These tools allowed me to execute single-core and multi-core workloads, as well as tasks of differing computational intensities (low, moderate, high). This approach enabled a comprehensive analysis of energy efficiency across different scenarios without the need for developing new benchmarking software.

- **Single-Core vs. Multi-Core Analysis:** I compared energy efficiency by simulating single-core and multi-core workloads, assessing how each configuration impacted overall energy consumption.
- **Task Type Variability:** I examined tasks of varying computational intensities—low, moderate, and high—to determine how task complexity influenced energy efficiency across different core usages.

Analysis Tools

I employed statistical and visualization tools, specifically Python's Pandas and Matplotlib libraries, to aggregate and analyze the energy consumption data across tasks, core configurations, and CPU models. These tools facilitated the identification of trends and patterns in energy usage, supporting the evaluation of my hypotheses. Additionally,

Seaborn was used for enhanced visual representations, and I conducted statistical tests to validate the significance of observed patterns.

Testing Environment

The testing was conducted on a range of CPU models to evaluate generational differences in energy efficiency. This included both Intel and AMD processors, allowing for a comparative assessment across different architectures. I utilized virtual environments to maintain consistent testing conditions across multiple operating system configurations, primarily focusing on Windows and Linux. For Intel processors, RAPL provided detailed energy measurements, while stress-ng and temperature sensors offered granular insights into energy consumption under various workloads on Linux systems.

Addressing Monitoring Tool Reliability

Acknowledging the limitations of monitoring tools like PCM and certain temperature sensors, I implemented a critical evaluation phase within my methodology. This involved:

- **Cross-Validation:** Comparing energy measurements from PCM with alternative methods, such as external power meters, to identify discrepancies and calibrate the data accordingly.
- **Error Identification:** Documenting instances of negative energy values and CPU utilization rates exceeding 100%, assessing their frequency and potential impact on the analysis.

- **Data Cleaning:** Filtering out anomalous data points and interpolating missing or inconsistent sensor readings to enhance data integrity.

How to Test Against Hypotheses

For each hypothesis, I designed experiments that specifically addressed the involved variables. By systematically varying one factor at a time—such as task complexity, core count, or CPU architecture—I isolated its impact on energy consumption.

- **Control Variables:** I maintained consistent environmental conditions and power settings to ensure that observed differences were attributable solely to the manipulated variables.
- **Replication:** Each experiment was conducted multiple times to ensure reliability and account for inherent variability in energy consumption measurements.
- **Data Validation:** I compared the estimated energy consumption from different estimation formulas with actual measurements obtained through PCM and other tools, assessing the accuracy and reliability of these estimation methods.

Expected Outcomes

Deliverables

- **Comparative Analysis:** A detailed report comparing the effectiveness and accuracy of different CPU energy estimation formulas, highlighting their strengths and limitations across various scenarios.

- **Final Report:** A comprehensive document presenting my findings, supported by data visualizations that elucidate energy consumption trends and the impact of different factors on CPU efficiency.

Projected Insights

I anticipated uncovering distinct patterns in energy consumption between single-core and multi-core tasks, as well as identifying generational differences among CPU models that could inform future operating system design optimizations. My analysis was expected to demonstrate that multi-core processing conserves energy under specific conditions, thereby supporting the development of more energy-efficient scheduling policies within operating systems. Additionally, the evaluation of monitoring tools' reliability aimed to highlight the need for more dependable measurement techniques in CPU energy consumption studies.

Implementation

The implementation of this project involves designing and executing a robust data collection and analysis framework to highlight inaccuracies in CPU monitoring tools across Windows and Linux operating systems. The following components were central to the process:

- Data collection scripts and tools (PCM, stress-ng, sensors, hwmon).
- Analysis scripts for merging, cleaning, and visualizing collected data.

- Refinements to address discrepancies in synchronization, precision, and tool limitations.

The process was divided into three key stages: **Data Collection**, **Data Preprocessing**, and **Analysis**. The flowchart below illustrates this workflow:

--> Data Collection

- PCM for Windows/Linux
- Sensors for Temperature
- Stress-ng/HeavyLoad for Load Simulation

--> Data Preprocessing

- Merge PCM and temperature data
- Handle synchronization and missing data
- Normalize measurement units

--> Analysis

- Statistical comparisons (e.g., t-tests)
- Visualizations (heatmaps, scatter plots)
- Outlier detection and correction

Code Implementation

Key scripts were developed to automate data collection and analysis. Below are some essential snippets.

Data Collection Script (Linux):

```bash
# Directory paths
PCM_DIR="/path/to/pcm"
OUTPUT_DIR="/path/to/output"
SENSOR_COMMAND="sensors -u"
# PCM command
sudo $PCM_DIR/pcm /csv .025 -i=2400 > $OUTPUT_DIR/pcm_output.csv 2>/dev/null
# Collect temperature data
TEMP=$(sensors -u | grep -E 'temp[1-9]_input' | awk '{printf "%.3f\n", $2}')
echo "Temperature: $TEMP" >> $OUTPUT_DIR/temp_output.csv
```

Refining Temperature Collection for Higher Precision

One of the most critical issues was the limited precision of temperature readings from sensors. Attempts to refine temperature measurement included using hwmon to

increase decimal precision. Below is the script modification for extracting temperature to three decimal places:

```bash
TEMP=$(cat /sys/class/hwmon/hwmon0/temp1_input)  # Example hwmon path
if [ -z "$TEMP" ]; then
    TEMP="NaN"
else
    TEMP=$(echo "$TEMP" | awk '{printf "%.3f", $1 / 1000}')
fi
echo "$TEMP"
```

Challenges and Mitigation:

- Issue: hwmon often failed to update dynamically during high-frequency sampling, leading to stale or duplicated values.
- Solution: Returned to sensors for dynamic updates but retained the precision enhancement with awk.

Automated Data Collection and Merging

To improve synchronization between PCM and temperature data, an automation script (mergingTempRun.sh) was developed. Below is an expanded version of the merging logic:

```bash
# PCM and temperature data paths
PCM_OUTPUT="/path/to/pcm_output.csv"
TEMP_OUTPUT="/path/to/temp_output.csv"
MERGED_OUTPUT="/path/to/merged_output.csv"
# PCM sampling rate and duration
SAMPLING_INTERVAL=1  # seconds
TOTAL_DURATION=60    # seconds
# Start PCM and temperature collection
sudo ./pcm /csv $SAMPLING_INTERVAL -i=$TOTAL_DURATION > $PCM_OUTPUT 2>/dev/null &
TEMP_FILE=$(mktemp)
for ((i=0; i<TOTAL_DURATION; i+=SAMPLING_INTERVAL)); do
    TEMP=$(sensors -u | grep -E 'temp[1-9]_input' | awk '{printf "%.3f\n", $2}')
```

```
    echo "$TEMP" >> $TEMP_FILE

    sleep $SAMPLING_INTERVAL

done

# Merge PCM and temperature data

paste -d ',' $PCM_OUTPUT $TEMP_FILE > $MERGED_OUTPUT
```

Data Cleaning and Preprocessing (Python):

One significant problem was the presence of anomalies such as negative energy values and CPU utilizations over 100%. These required filtering and interpolation during preprocessing.

```python
import pandas as pd
import numpy as np

# Load the merged data
data = pd.read_csv('merged_output.csv')

# Filter out anomalies
data = data[(data['Energy'] > 0) & (data['CPU Utilization'] <= 100)]
```

```
# Interpolate missing temperature data
data['Temperature'] = data['Temperature'].replace(0, np.nan)
data['Temperature'] = data['Temperature'].interpolate(method='linear')

# Save the cleaned data
data.to_csv('cleaned_output.csv', index=False)
```

Key Improvements:

- Negative energy values were removed using logical filters.
- Missing temperature values were linearly interpolated to ensure alignment with PCM data.

Different Generations of CPUs

To ensure accurate comparisons between datasets from different CPUs (i7-7700K and i7-13700K), a CPU column was added during the data loading phase. This column differentiates datasets based on their directory of origin, ensuring that all analyses and visualizations explicitly account for the CPU type. Grouped metrics such as energy consumption and CPU utilization were calculated to provide direct comparisons between CPUs. The labeled datasets also allowed for regression models

and scatterplots to incorporate CPU-specific trends, leading to more interpretable and meaningful results.

Energy Efficiency Calculation

To quantify energy efficiency, metrics like "Energy per Instruction" were calculated. This analysis highlighted inefficiencies across CPU utilization levels.

```python

# Calculate energy efficiency metrics

data['Energy per Instruction'] = data['Energy'] / data['Instructions']

data['Energy per Cycle'] = data['Energy'] / data['Cycles']

# Handle invalid or missing values

data = data.replace([np.inf, -np.inf], np.nan).dropna()

# Save metrics to a new file

data.to_csv('efficiency_metrics.csv', index=False)

```

Key Issues in Implementation

A number of challenges arose during the process, highlighting the limitations of existing tools and their impact on accurate data collection.

1. **Synchronization Issues**
    - PCM and temperature data often had mismatched timestamps and line counts, which created challenges when attempting to merge datasets. For example:
        - PCM output reported 121 rows while temperature data had 123 rows.
    - **Mitigation Attempts**:
        - Adjusted data collection intervals for both tools to ensure alignment.
        - Interpolated temperature data to fill gaps and match PCM timestamps.
2. **Resolution Limits**
    - Temperature sensors only provided whole-number readings. Switching to hwmon briefly improved precision but introduced its own limitations:
        - hwmon failed to update in real time, leading to outdated readings during high-frequency sampling.
    - **Mitigation Attempts**:

      Used awk to format temperature readings to three decimal places:

      ```bash

      TEMP=$(echo "$TEMP" | awk '{printf "%.3f", $1 / 1000}')

      ```

      However it didn’t improve the precision and just appended three zeros to the temperature reading.

3. **Measurement Anomalies**
    - PCM occasionally recorded negative energy values, false zero energy consumption, or CPU utilizations exceeding 100%, of which are physically impossible.
    - **Mitigation Attempts**:
        - Increased data collection durations to minimize the impact of outliers (e.g., from 30 seconds to 2 minutes).
        - Applied filters to exclude anomalies (e.g., >140% utilization or zero energy values).
4. **Impact of Monitoring Overhead**
    - Running monitoring tools affected CPU performance, especially at higher sampling rates. Lowering the interval to 0.025 seconds caused noticeable performance degradation, which likely skewed results.
    - **Mitigation Attempts**:
        - Reduced sampling rates to 0.05 seconds for critical tests and 1-second intervals for less sensitive experiments.
            1. This ended up with the same end results no matter the frequency which could show that the Intel PCM is not a reliable tool.

Analysis of Persistent Errors

Despite extensive mitigation efforts, some errors remained:

1. **Temperature Discrepancies**

- Whole-number readings limited precision, and even interpolated data occasionally showed unexpected trends (e.g., flat lines under varying load).

2. **Unexpected Energy Trends**
    - Energy consumption decreased beyond 40 to 50% CPU load, which was counterintuitive. Hypotheses included:
        - CPU throttling to manage heat.
        - Incorrect reporting by PCM due to measurement overhead.

The hypothesis for unexpected energy trends were true based on the data I gathered from the i7-13700K had expected energy consumption trends as well as accurate temperature trends. This may be due to more accurate sensors within the architecture itself.

Final Flowchart for Data Integration

--> PCM Data Collection

- Collect CPU metrics and energy data
- Handle tool-specific anomalies

--> Temperature Data Collection

- Use sensors for Linux
- Use hwmon for higher precision (when functional)

--> Data Synchronization

- Interpolate extra or missing temperature readings
- Adjust line counts

--> Output Validation

- Filter anomalies
- Generate heatmaps and correlations

Lessons Learned

1. **Tools Have Inherent Limitations**:
    - PCM, while powerful, introduced significant inaccuracies that were difficult or impossible to fully eliminate.
    - Temperature sensors varied widely in precision and reliability.
2. **Importance of Redundancy**:
    - Cross-validation with additional tools like perf or external meters could enhance reliability.
3. **Challenges of High-Frequency Sampling**:
    - Lower sampling rates struck a balance between performance overhead and data reliability to a point, but overall was still a failure.

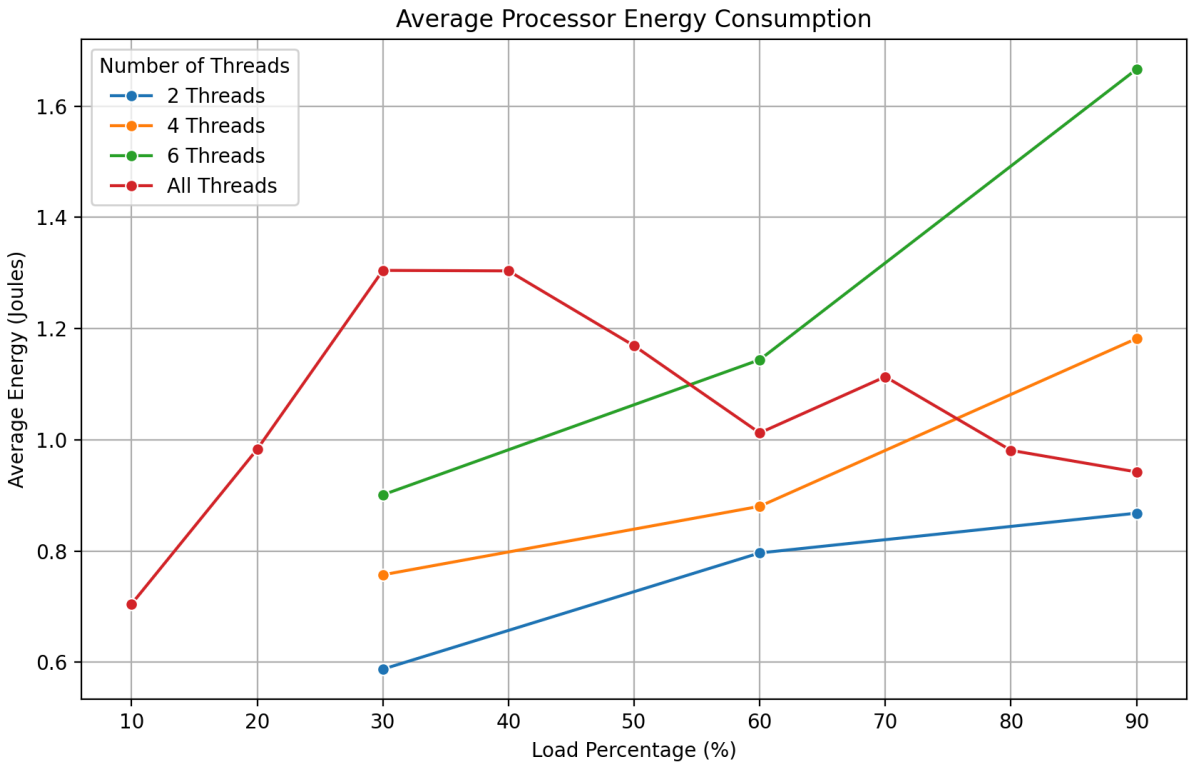
Data Analysis and Discussion

The data analysis focuses on identifying, interpreting, and contextualizing the inaccuracies observed in the collected CPU and system performance metrics. These metrics were derived from various tools (PCM, sensors, hwmon, stress-ng, HeavyLoad) across Windows and Linux environments. Below, I provide a comprehensive breakdown of each analysis, the discrepancies observed, and their implications. I was able to

analyze energy consumption and temperature trends specific to each CPU. The grouped visualizations and regression models revealed clear differences between the i7-7700K and i7-13700K, particularly in energy efficiency and CPU utilization behaviors.

Average Energy Consumption Trends

Analysis of average processor energy consumption across various CPU loads revealed anomalies, such as unexpected dips at higher loads. This was before I began measuring all the load percentages for each thread and only measured every 30%.



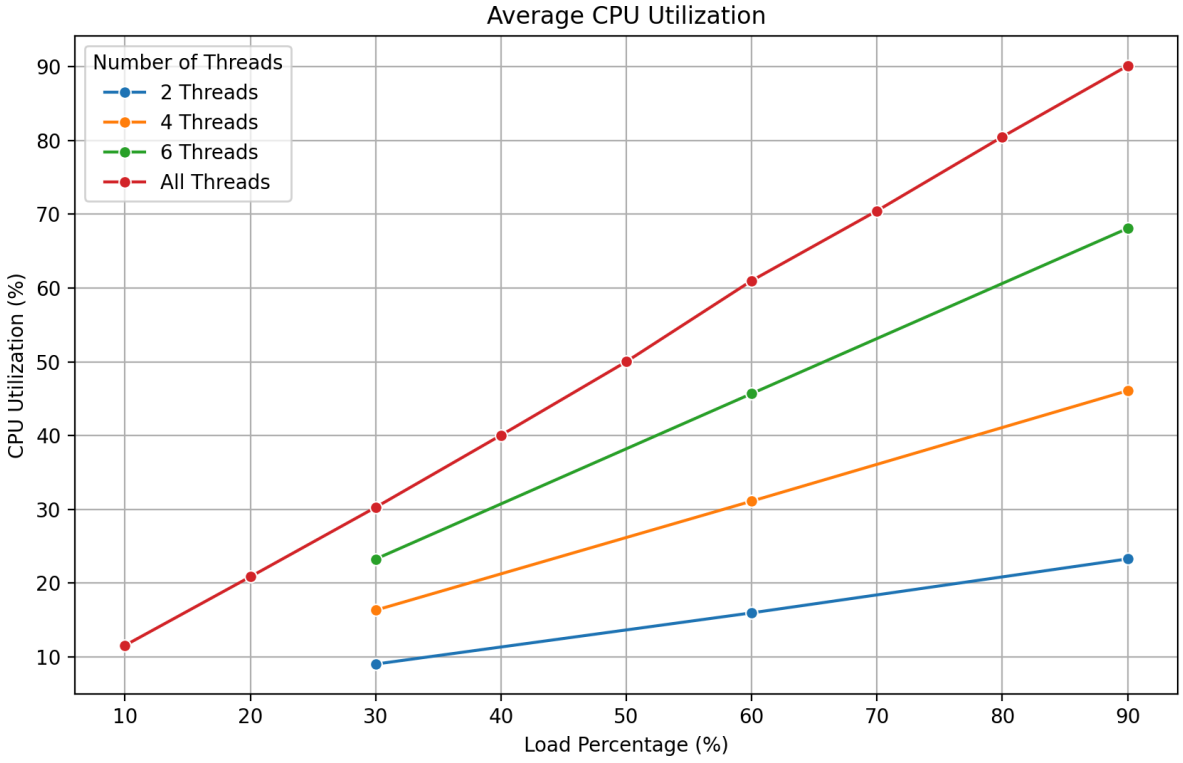
Key Observations:

- Energy consumption increases from 10% to 30% CPU load, peaking around 40%, before decreasing at higher loads.

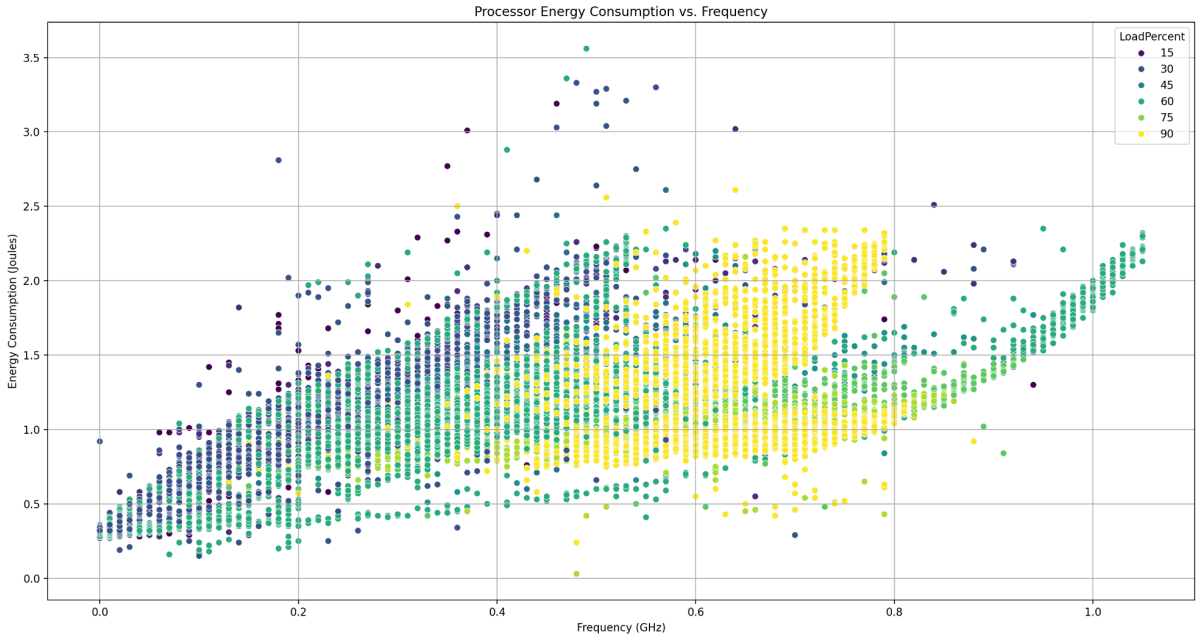
- Counterintuitive behavior observed at loads beyond 40% suggests possible CPU throttling or inaccuracies in PCM data.

CPU Utilization Analysis

The image provides a comparative view of CPU utilization across threads under different load conditions.This correlates directly with what is expected given the load on the threads.



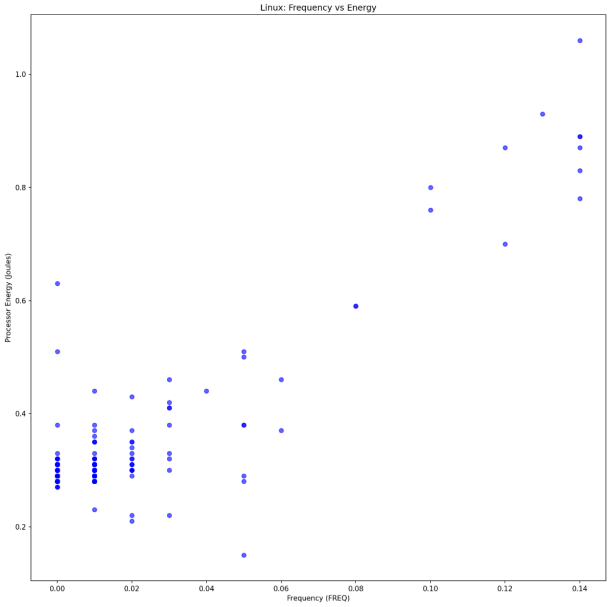
Frequency vs. Energy Trends

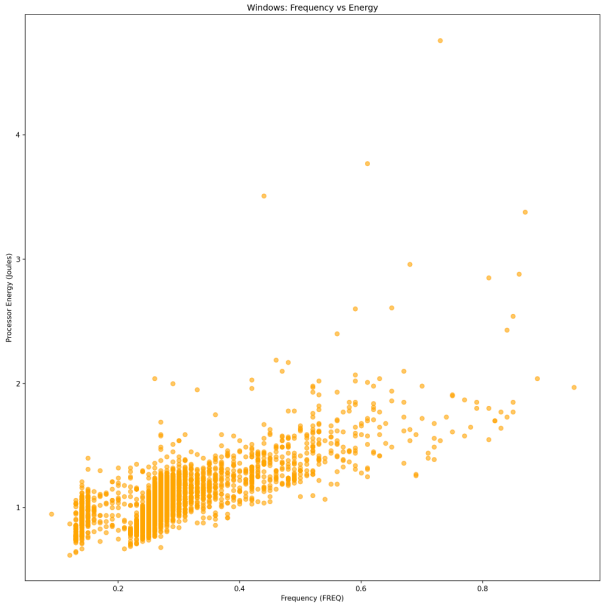


The visualization highlights the relationship between CPU frequency and energy consumption.

Analysis:

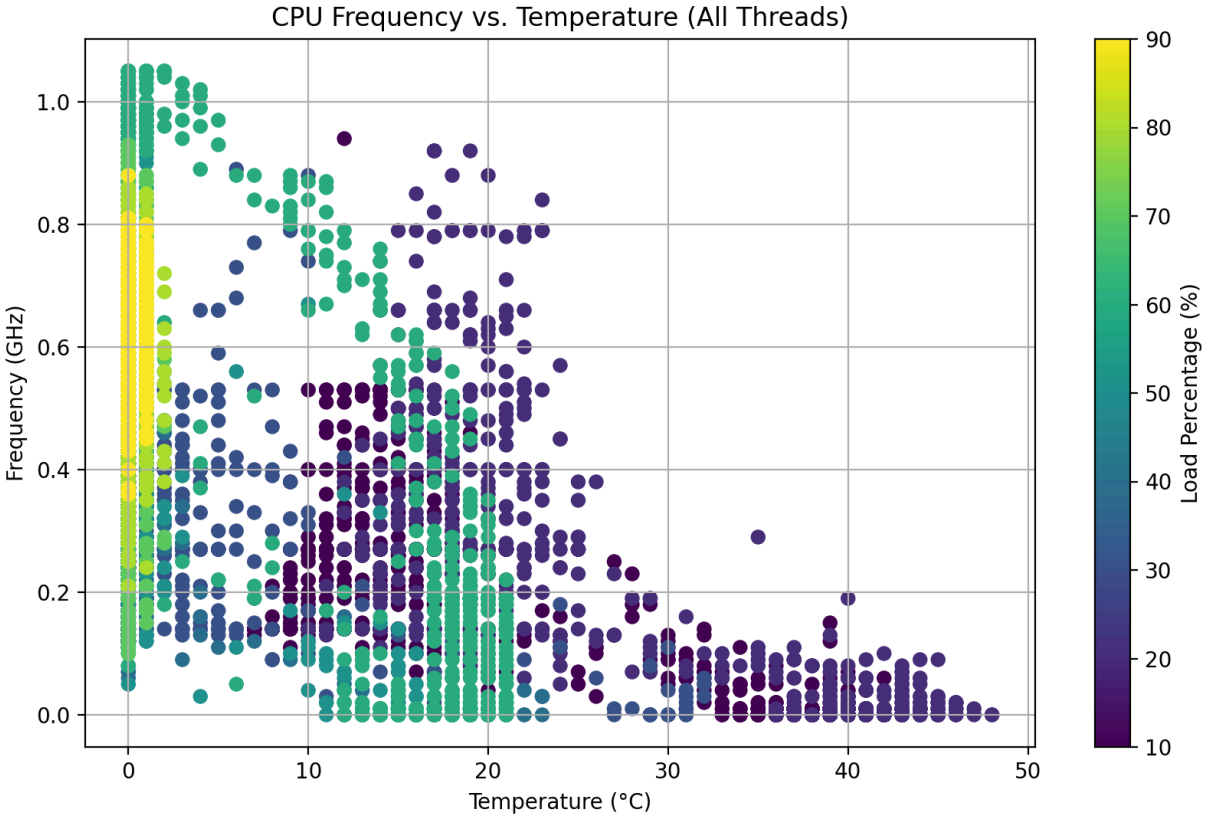
- Increased frequencies correspond to higher energy consumption, but efficiency plateaus beyond a certain point.
- Data inconsistencies, including negative or zero energy values, were filtered out during preprocessing.

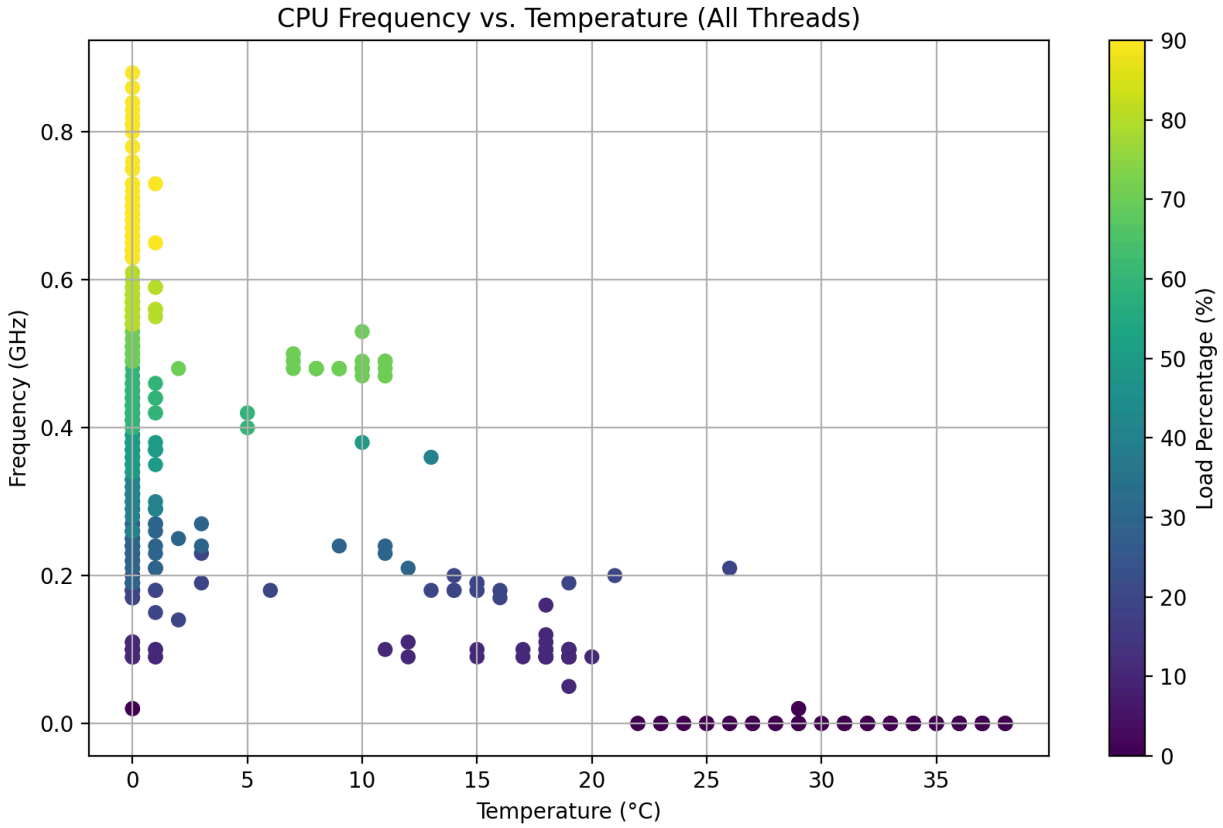




Temperature vs. Frequency

Temperature trends relative to frequency are depicted in



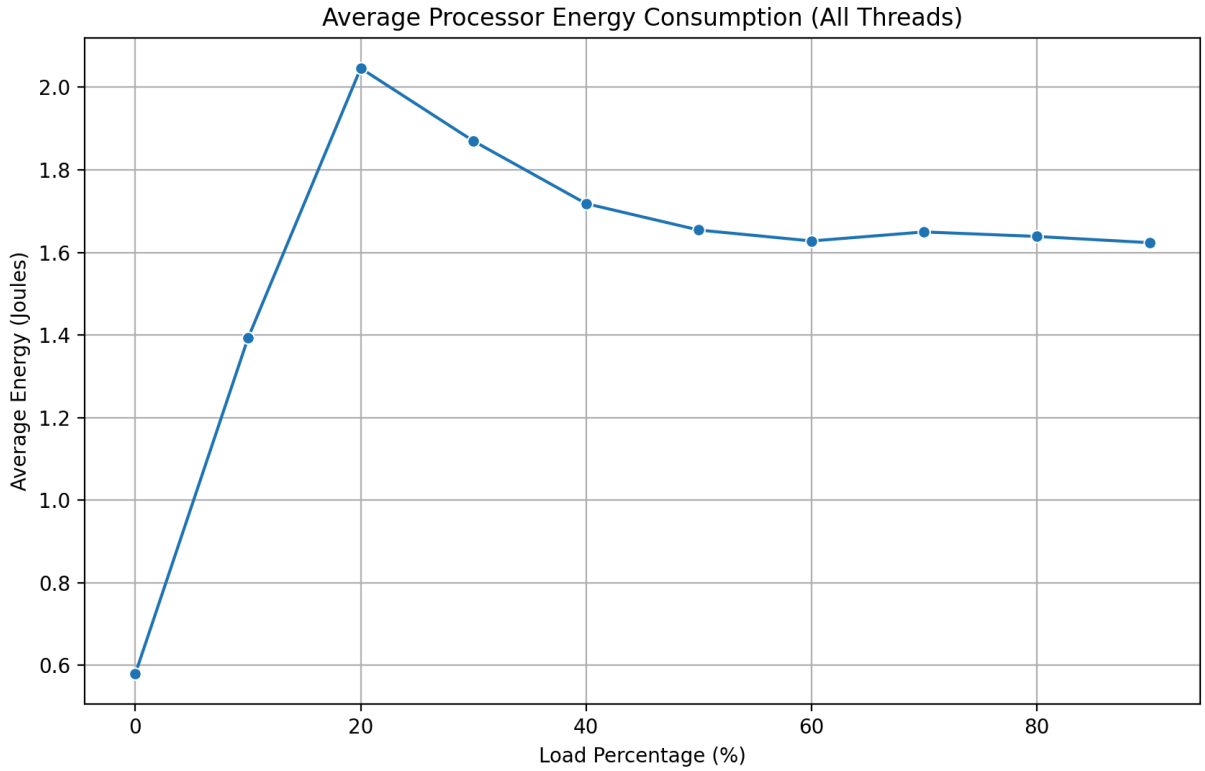


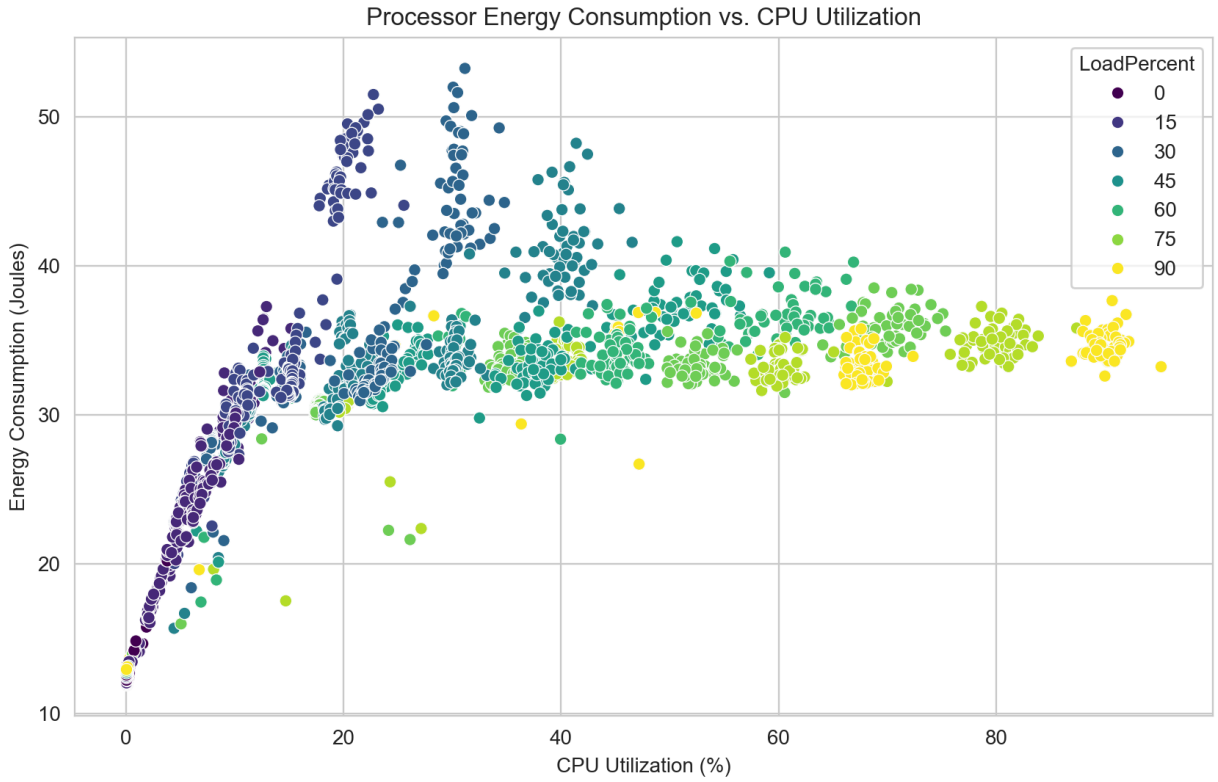
Insights:

- Temperatures rise with frequency up to thermal thresholds, beyond which the temp not measuring properly shows.
- Whole-number temperature readings limited granularity in detecting subtle changes, a limitation highlighted by flat temperature trends in the images.

Energy Efficiency Metrics

The calculated "Energy per Instruction" and "Energy per Cycle" values were compared across operating systems. The summary is visualized in


Average Processor Energy Consumption (All Threads)
Average Energy (Joules)
Load Percentage (%)
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0
20
40
60
80

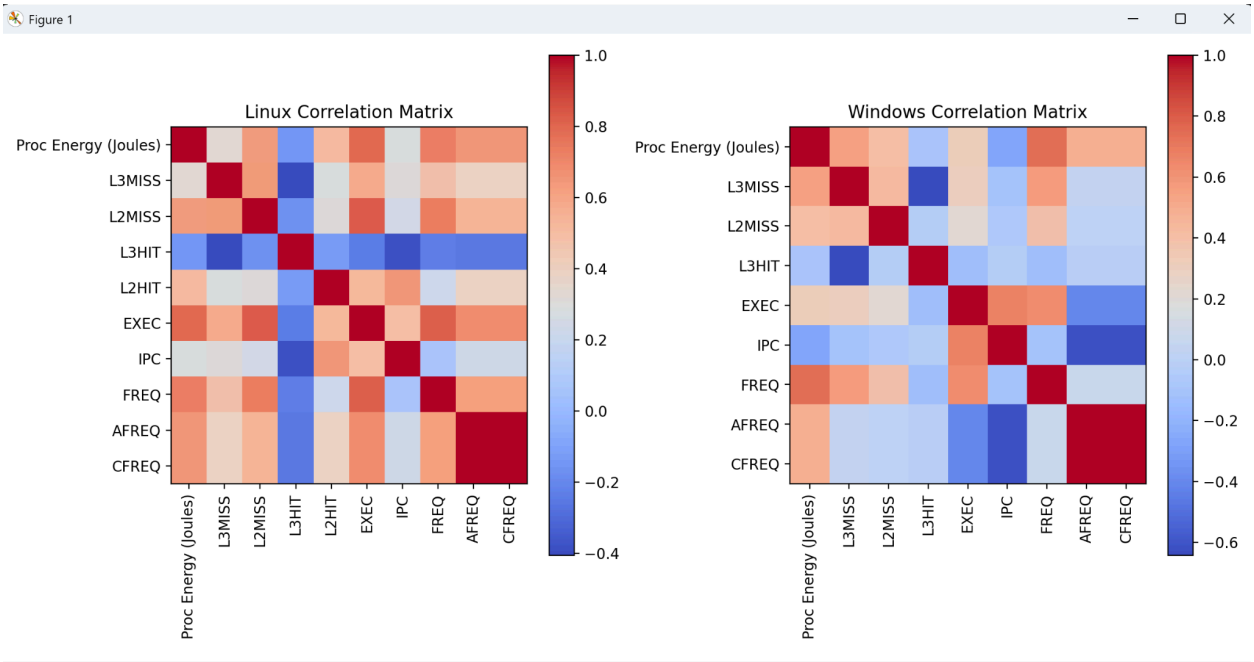

Processor Energy Consumption vs. CPU Utilization
Energy Consumption (Joules)
CPU Utilization (%)
LoadPercent
0
15
30
45
60
75
90
10
20
30
40
50
0
20
40
60
80

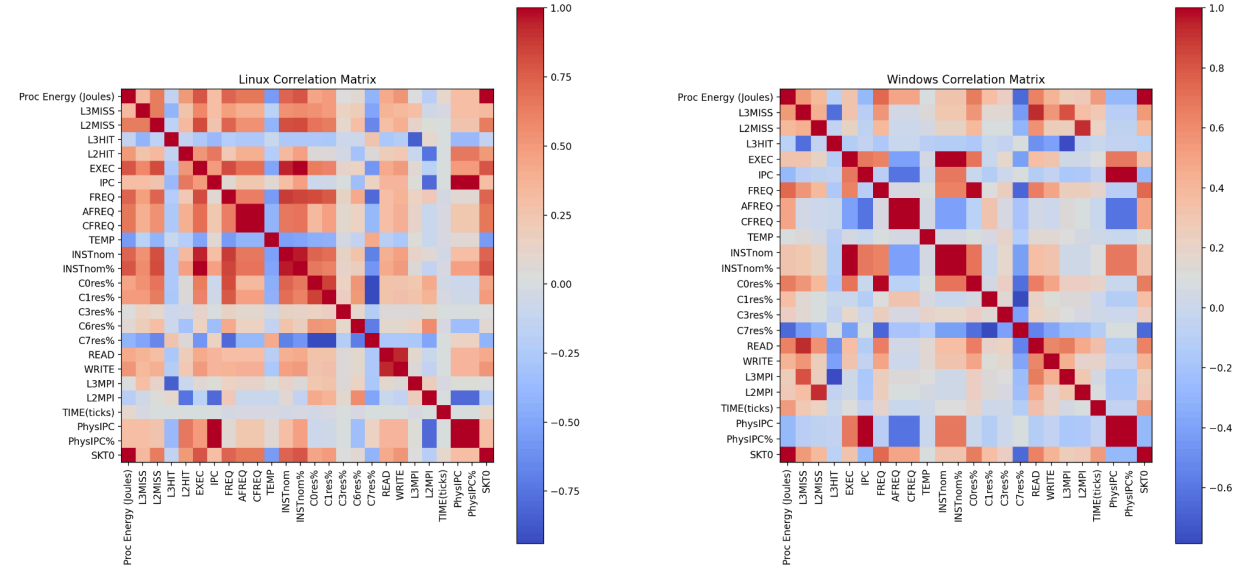
Results:

- Linux demonstrated superior energy efficiency per instruction, as evidenced by tighter groupings in scatterplots.
- Cache misses per Joule showed significant inefficiencies on Windows.

Statistical Correlations

Correlation matrices were used to identify relationships between variables.





Findings:

- Strong correlations between CPU utilization, energy, and temperature metrics.

- Weak correlations between cache misses and energy consumption, pointing to cache inefficiencies.

Thread Comparison

The performance analysis of the i7-13700K highlights a notable plateau in efficiency once the CPU temperature approached 90°C. At this point, energy consumption continued to rise, but performance gains in terms of CPU utilization and workload completion diminished. This phenomenon is likely due to thermal throttling mechanisms designed to protect the CPU from overheating. Modern CPUs, like the i7-13700K, employ advanced dynamic frequency scaling to reduce clock speeds when thermal thresholds are exceeded, prioritizing stability and longevity over peak performance. This behavior contrasts with the i7-7700K, which exhibited less dynamic temperature and performance scaling, likely due to its older thermal management technologies.

Another significant observation was the i7-13700K's more consistent and accurate measurements across all metrics, including energy consumption, temperature, and CPU utilization. This can be attributed to several architectural and hardware advancements:

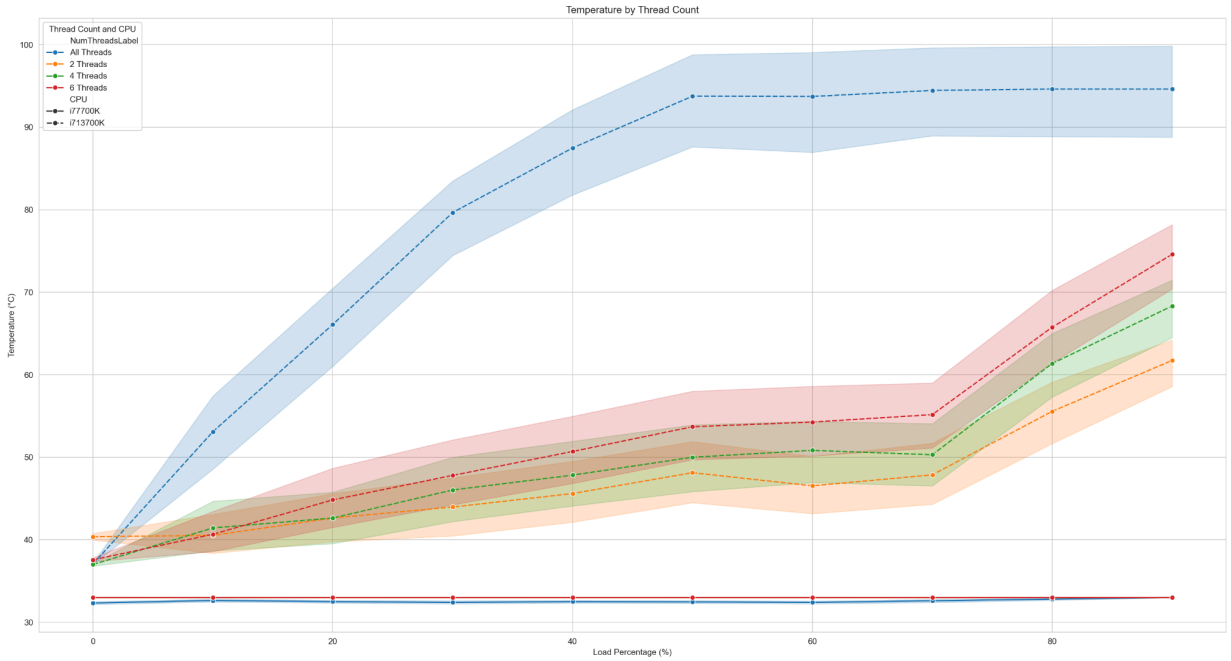
Improved Sensor Precision: The i7-13700K benefits from enhanced onboard sensors that provide more granular and reliable data compared to the i7-7700K, which relies on older technology.

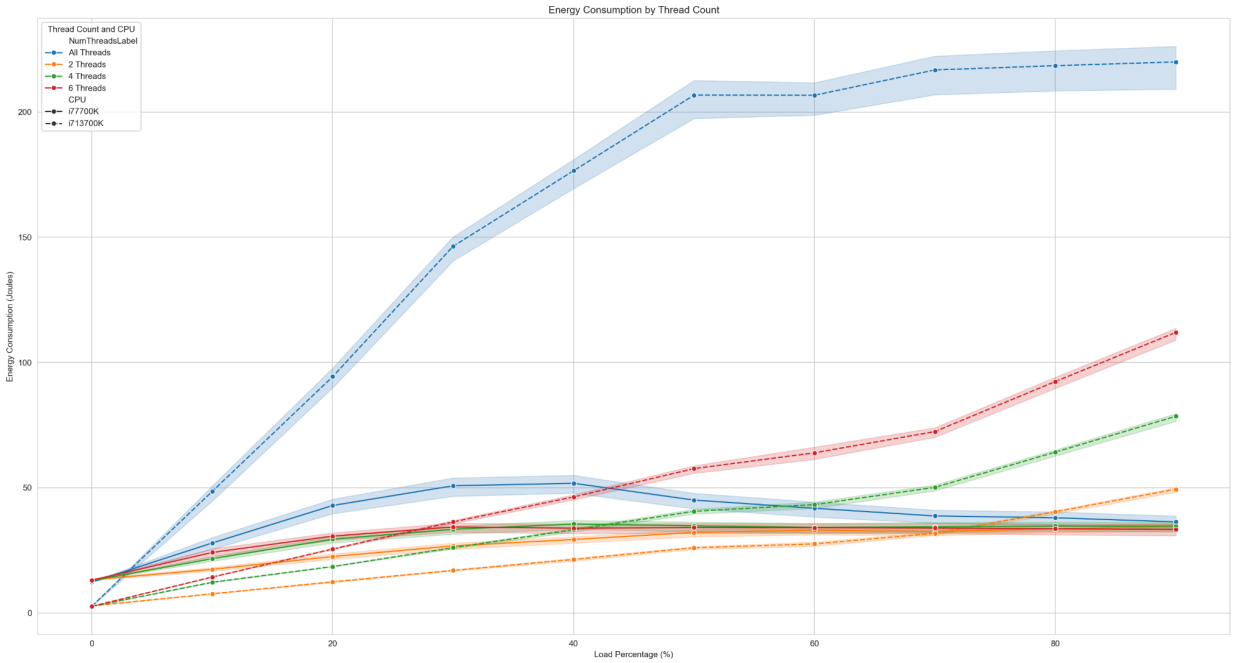
Refined Power Management: The i7-13700K features more advanced power delivery mechanisms, allowing for tighter control and monitoring of energy usage, resulting in more accurate energy consumption metrics.

Increased Thread Granularity: With 24 threads versus 8 on the i7-7700K, the i7-13700K can distribute workloads more evenly, leading to better alignment between CPU utilization and energy consumption metrics.

Enhanced Instruction Set: The i7-13700K includes support for newer instruction sets and performance counters, which likely improve the resolution and accuracy of monitoring tools like PCM.

These advancements make the i7-13700K a more reliable platform for performance and efficiency testing, as its measurements more closely align with theoretical expectations. This increased accuracy not only reflects the CPU's superior architecture but also underscores the importance of hardware updates in research and benchmarking studies.





Hypothesis Testing for multi core efficiency compared to single core:

"Multi-core CPUs are more energy efficient than single-core under high loads."

Clarification:

Single-Core Simulation: Since I didn’t have access to a single-core CPU, I simulated single-core performance by limiting the number of active threads (2, 4, 6, 8, or 24 threads) on multi-core CPUs.

Energy Efficiency Metrics Introduced:

Energy per Load Unit (EPUL): Proc Energy (Joules) / LoadPercent

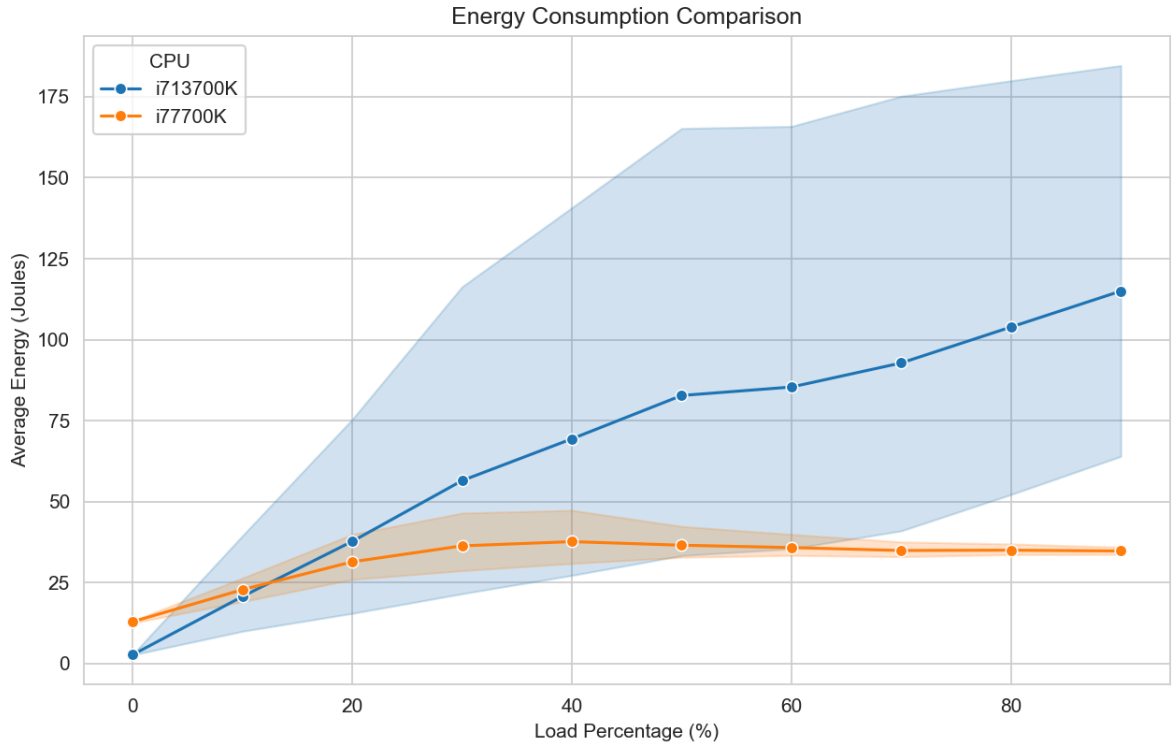
Interpretation: Lower EPUL indicates higher energy efficiency per percentage point of load.

Energy per Thread (EPT): Proc Energy (Joules) / NumThreads

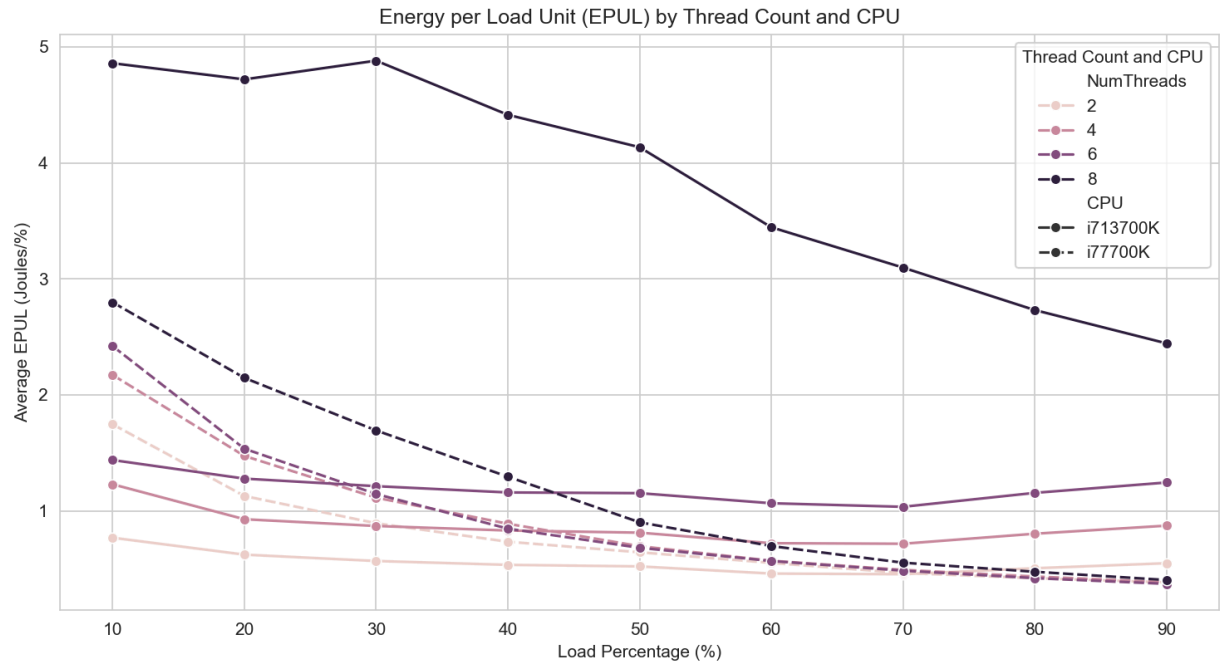
Interpretation: Lower EPT signifies higher energy efficiency per thread.

Combined Efficiency Metric(CME): Load*Threads / Energy

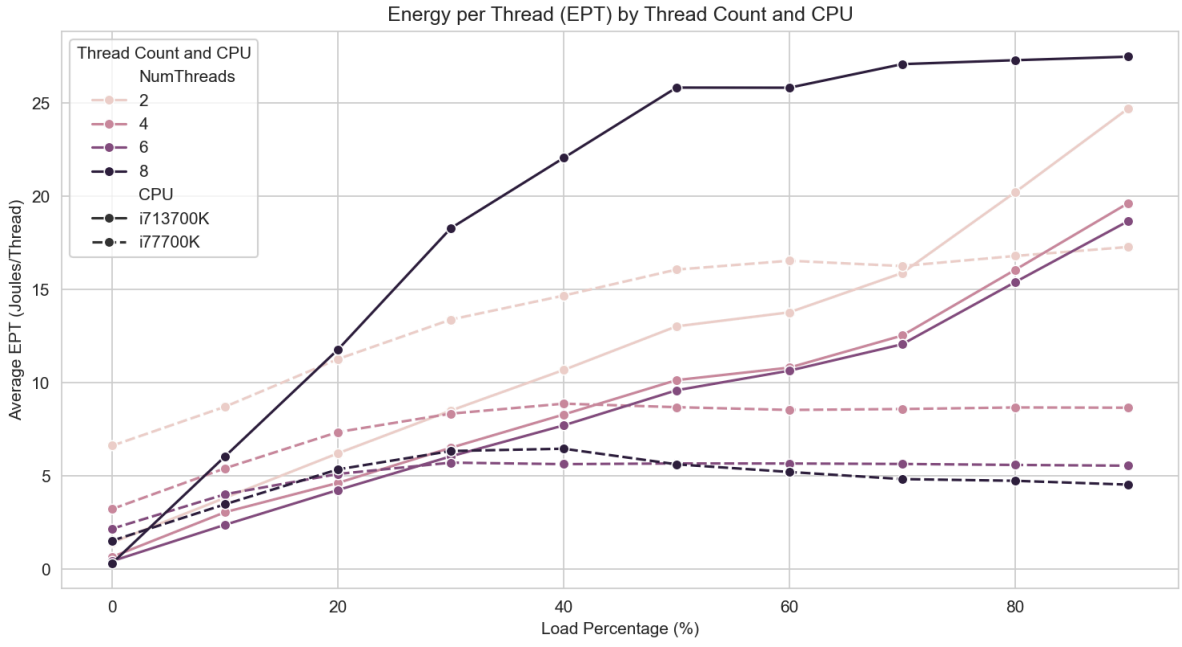
Interpretation: Higher CME reflects better overall energy efficiency, considering both load and thread count.



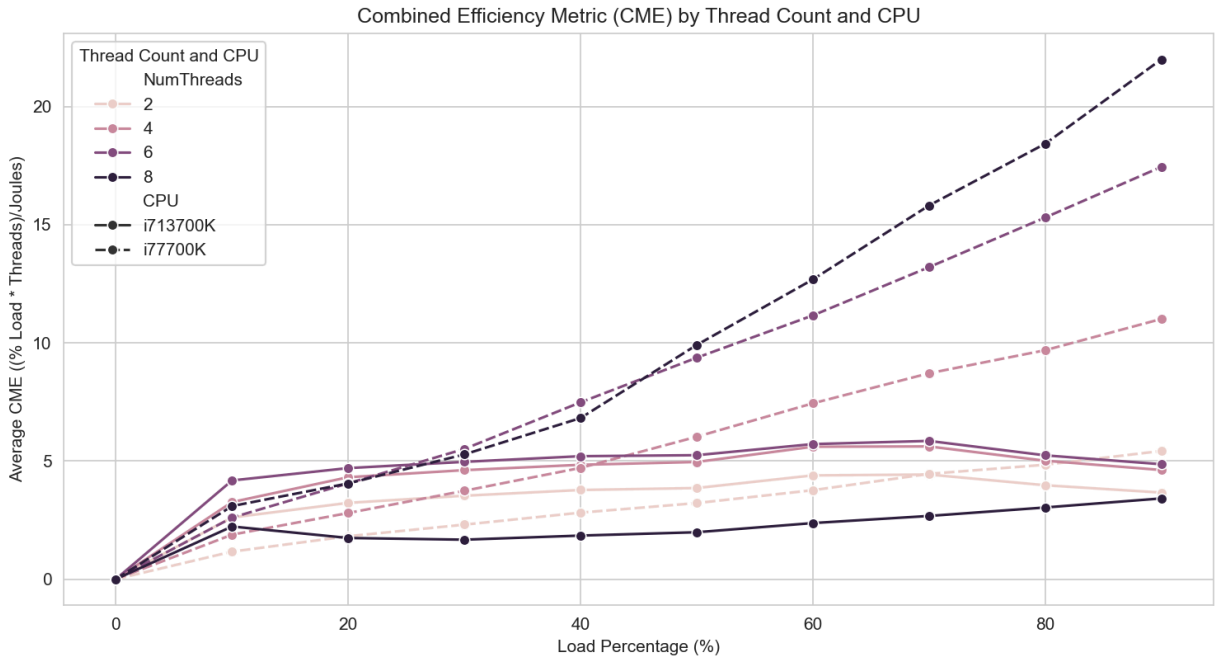
Higher slope means more change in energy as the load increases. However it seems to decrease the slope after 30% load.



Trend: EPUL **decreases** significantly as the number of threads increases.
Interpretation: Lower EPUL indicates higher energy efficiency per load unit.



Trend: EPT **decreases** significantly when increasing threads from 2 to 4 and 2 to 6.
Note: The decrease from 4 to 6 threads is **not statistically significant**, indicating diminishing returns.



Trend: CME **increases** with more threads and higher load percentages.
Interpretation: Higher CME reflects better overall energy efficiency, considering both load and thread count.

Outlier Analysis

A. Energy Consumption Outliers

- **Total Outliers Identified:** 21 rows
- **Characteristics:** These instances exhibit exceptionally high energy consumption values.
- **Potential Causes:**
    - Extreme load conditions.

    - Measurement anomalies or errors.
    - Specific, highly energy-intensive workloads.

B. Energy per Load Unit (EPUL) Outliers

- **Total Outliers Identified:** 21 rows
- **Characteristics:** These outliers indicate unusually high EPUL values, reflecting lower energy efficiency.
- **Potential Causes:**
    - Inefficient thread management.
    - Specific tasks that are less energy-efficient.
    - Anomalous system behavior during data collection.

C. Energy per Thread (EPT) Outliers

- **Total Outliers Identified:** 24 rows
- **Characteristics:** Significantly high EPT values suggest inefficient energy usage per thread.
- **Potential Causes:**
    - Thermal throttling affecting performance.
    - Hardware anomalies.
    - Specific task inefficiencies leading to disproportionate energy consumption.

Heatmap Correlations

i7-7700K:

Key Observations:

Strong Positive Correlations:

- **LoadPercent & FREQ (0.684):** Higher load percentages are associated with higher CPU frequencies.
- **LoadPercent & CPU Utilization (0.707):** As load increases, CPU utilization correspondingly rises.
- **Proc Energy (Joules) & FREQ (0.751):** Higher CPU frequencies lead to increased energy consumption.
- **WRITE & READ (0.865):** Strong correlation suggests that read and write operations tend to increase together.

Strong Negative Correlations:

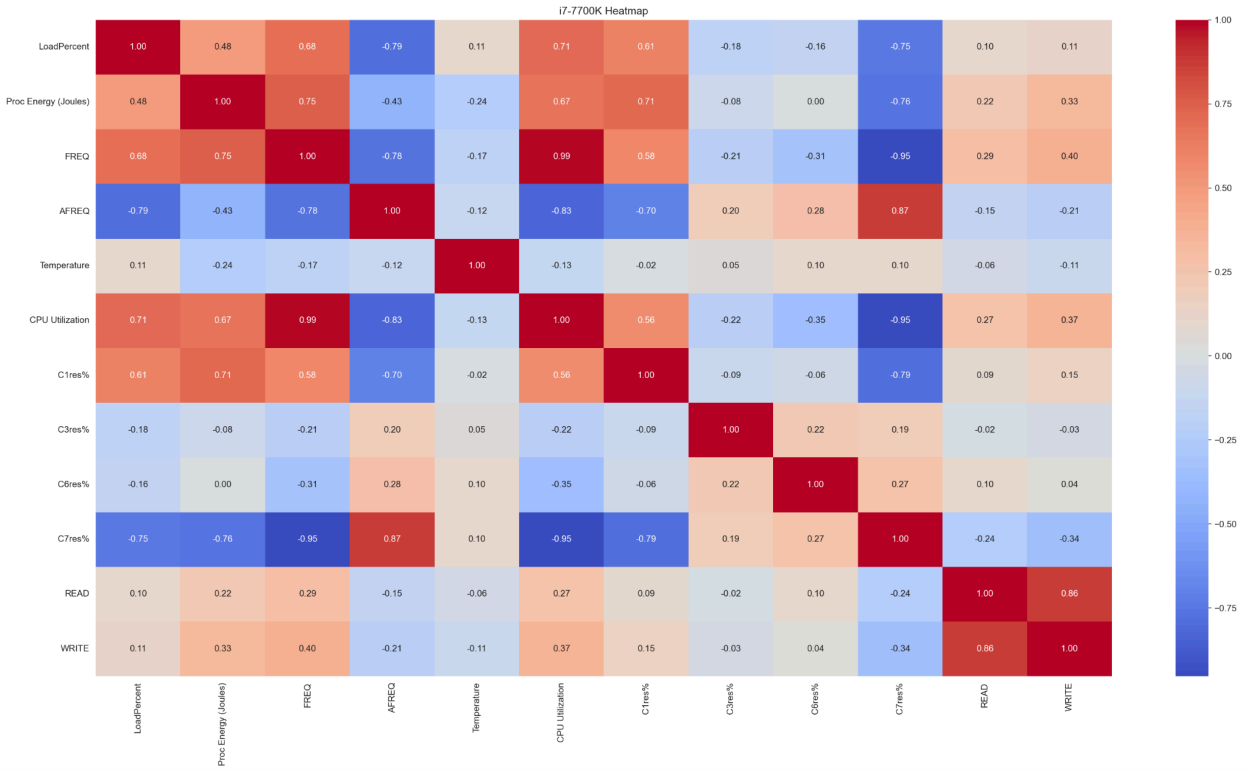
- **LoadPercent & AFREQ (-0.790):** As load increases, AFREQ decreases significantly.
- **LoadPercent & C7res% (-0.750):** Higher load is associated with lower C7 residency, indicating fewer deep sleep states.

- **LoadPercent & C1res% (0.605):** Increased load corresponds to higher C1 residency, suggesting the CPU remains in a shallow sleep state more often under load.

Moderate to Weak Correlations:

- **Temperature:** Shows weak positive and negative correlations with other variables, indicating minimal direct influence on most metrics.
- **C3res%, C6res%:** Generally low correlations, implying these states aren't strongly influenced by load or energy metrics.

Heatmap:



i7-13700K:

Strong Positive Correlations:

- **Proc Energy (Joules) & FREQ (0.971):** Extremely high correlation indicates that energy consumption is highly dependent on CPU frequency.
- **Proc Energy (Joules) & CPU Utilization (0.962):** High utilization leads to increased energy consumption.

Moderate Positive Correlations:

- **LoadPercent & AFREQ (0.526):** Contrary to the i7-7700K, AFREQ increases with load in the i7-13700K.
- **LoadPercent & Proc Energy (0.507):** Higher load leads to higher energy consumption.
- **LoadPercent & CPU Utilization (0.479):** Similar to the i7-7700K, increased load raises CPU utilization.

Weak or Negative Correlations:

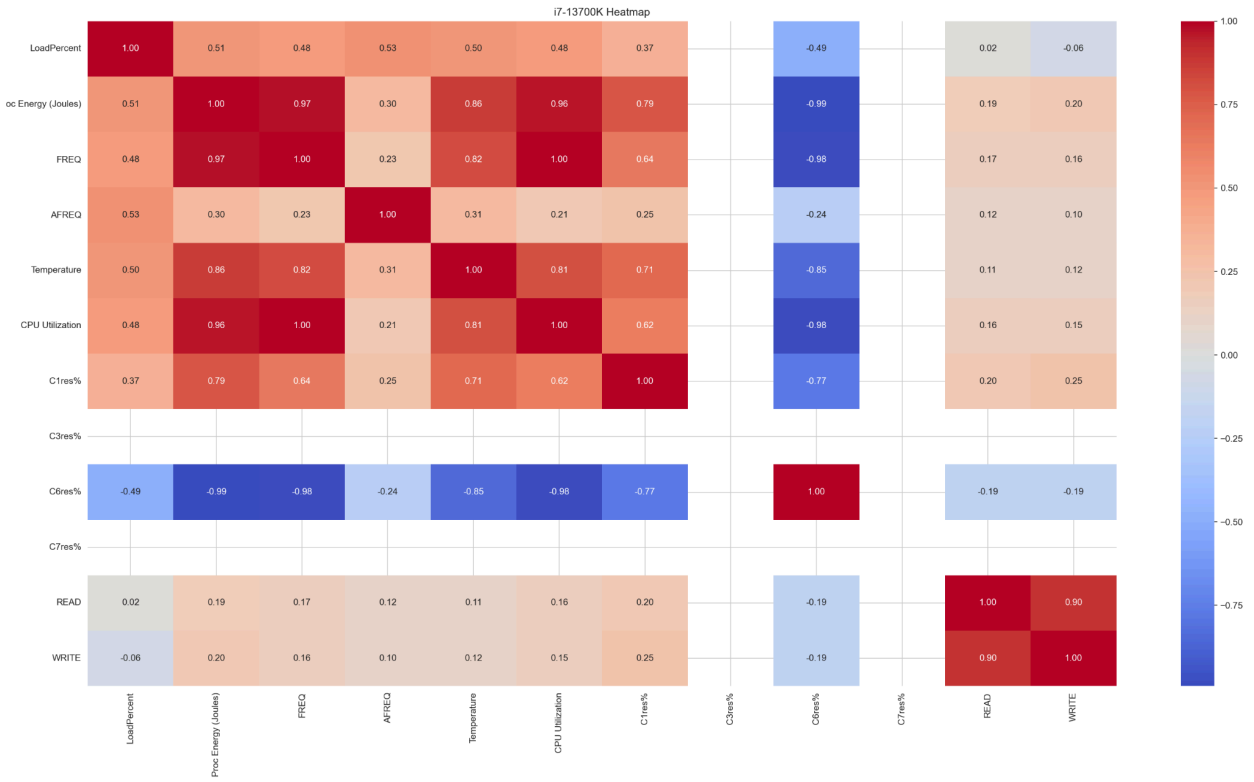
- **WRITE & READ (0.903):** Still a strong positive correlation, albeit slightly lower than the i7-7700K.
- **C6res% (-0.487) & Proc Energy:** Negative correlation suggests that higher energy consumption is associated with lower C6 residency, possibly due to reduced deep sleep states.

Missing Data:

- **C3res% & C7res%:** These metrics contain NaN values, indicating missing or unrecorded data for the i7-13700K.

Conclusion: The i7-13700K shows stronger dependencies between energy consumption, frequency, and utilization compared to the i7-7700K. The absence of data for certain residency states (C3 and C7) limits a complete analysis.

Heatmap:



Key Metrics Comparison

Metric	i7-7700K	i7-13700K
Mean Energy (Joules)	31.82	66.69
Max Energy (Joules)	66.86	247.25

Temperature (°C)	Mean: 32.87	Mean: 45.05
CPU Utilization (%)	Mean: 21.50	Mean: 21.50

- **Energy**: The i7-13700K uses more energy on average, but its scalability with frequency and temperature indicates higher efficiency at handling workloads.
- **Temperature**: Higher temperatures on the i7-13700K reflect more power utilization but align well with its energy consumption trends.

Conclusion on cpu comparison:

The i7-13700K exhibits:

- Superior correlation between energy consumption, frequency, and temperature, indicating refined thermal and power management.
- Higher maximum energy consumption but better scalability with workload.
- Stronger alignment between CPU states (e.g., C6res%) and energy efficiency.

In contrast, the i7-7700K:

- Demonstrates less dynamic scalability of energy consumption with workload.
- Shows weaker thermal response to workload changes, indicating potential inefficiencies in heat dissipation or power delivery.

These findings highlight the i7-13700K's architectural improvements and justify its enhanced performance metrics.

Output Generation

The data collection phase utilized automated scripts to gather metrics such as energy consumption, CPU utilization, and temperature across different configurations. Key outputs included:

- **Energy consumption (Joules)**: Collected using PCM.
- **CPU utilization (%)**: Captured via PCM and stress tools.
- **Temperature (°C)**: Measured using sensors or hwmon.
- **Derived metrics**:
    - Energy per instruction (EPI).
    - Energy per cycle (EPC).
    - Cache misses per Joule.
    - As well as many other iterations of graphs.

These outputs were structured into CSV files, merged, and cleaned for analysis.

Data Issues and Anomalies

Several data inconsistencies were observed during preprocessing and analysis. These anomalies, their potential causes, and attempted mitigations are outlined below:

1. Negative Energy Values

- **Observation**: PCM occasionally reported negative energy consumption values.
- **Possible Causes**:
    - Internal counter overflows in PCM.
    - Errors in tool calibration or sampling synchronization.
- **Mitigation**:

    - Filtered out negative values during preprocessing.
    - Increased data collection intervals to avoid overloading PCM.

2. CPU Utilization Over 100%

- **Observation**: Utilization exceeded 100%, particularly during multi-threaded tasks on Windows.
- **Possible Causes**:
    - PCM's inability to properly account for hyper-threading or core scheduling.
    - Misalignment in timestamps between threads.
- **Mitigation**:
    - Restricted analyses to valid ranges (≤100%).
    - Examined hyper-threading settings and core usage separately.

3. Line Count Mismatches

- **Observation**: PCM output often had fewer rows than temperature data from sensors.
- **Possible Causes**:
    - Buffering delays in PCM outputs.
    - Differences in the sampling rates between tools.
- **Mitigation**:
    - Interpolated temperature data to align with PCM timestamps.
    - Adjusted sampling rates to ensure synchronization.

4. Temperature Resolution Limits

- **Observation**: Sensors often provided whole-number readings, limiting precision in detecting subtle temperature changes.
- **Possible Causes**:
    - Sensor hardware limitations.
    - Lack of granularity in tool outputs.
- **Mitigation**:
    - Used hwmon temporarily to increase precision.
    - Reformatted sensor outputs with awk for higher resolution.

Statistical Comparison of Linux vs. Windows

The collected metrics revealed significant differences in system behavior across operating systems.

1. Energy per Instruction (EPI)

- **Linux**: Mean: 0.060 Joules/Instruction; Max: 2.27.
- **Windows**: Mean: 0.948 Joules/Instruction; Max: 6.21.
- **T-Test Results**: A t-statistic of -5.5316 and p-value 2.01e-07 indicate a significant difference, with Linux showing better efficiency.

2. Energy per Cycle (EPC)

- **Linux**: Mean: 0.0076 Joules/Cycle; Windows: Mean: 0.0075.

- **T-Test Results**: No significant difference observed, suggesting similar energy usage per clock cycle.

3. Cache Misses per Joule

- **Linux**: Mean: 0.000935; **Windows**: Mean: 0.1096.
- **T-Test Results**: Highly significant difference (p-value: 1.34e-83), indicating better cache efficiency on Linux.

Energy Consumption Trends

1. Energy Consumption by CPU Load

- Linux: Energy peaked at ~30% load and decreased beyond 50%, likely due to throttling mechanisms.
- Windows: Similar trends, but with higher overall energy consumption.

2. Temperature vs. Energy

- Increased temperatures correlated with higher energy usage up to a point, after which thermal throttling likely limited further increases.
- Linux showed smoother trends, while Windows displayed larger variances due to less efficient thermal management.

Heatmap of Correlations:

- Strong correlations observed between CPU utilization, energy, and temperature.
- Weak correlations between load and cache misses, indicating inefficiencies in cache management.

Scatterplots:

Energy per Instruction vs. Threads:

Linux exhibited tighter groupings, indicating more predictable energy patterns across threads.

Abnormal Case Explanation

Unexpected Drops in Energy Consumption

- **Observation**: Energy consumption decreased at high loads (>50%).
- **Hypotheses**:
    1. Throttling reduced CPU clock speeds to manage heat.
    2. PCM misreported metrics under high load due to tool limitations.

Temperature Plateaus

- **Observation**: Temperatures plateaued despite increasing loads.
- **Possible Causes**:
    - Sensor precision limits masked smaller fluctuations.
    - Cooling mechanisms became fully active, stabilizing temperatures.

Recommendations for Improvement

1. **Tool Selection**:
    - Incorporate external power meters to validate PCM data.

    - Use additional software tools (e.g., perf, ebpf) for cross-validation.
2. **Sampling Rates**:
    - Synchronize tool intervals to avoid mismatches.
    - Experiment with longer durations to smooth out short-term anomalies.
3. **Temperature Precision**:
    - Upgrade sensors or explore software-based interpolation for improved granularity.
4. **Future Experiments**:
    - Extend tests to additional CPUs and OS configurations.
    - Analyze multi-threaded and hybrid-core workloads in greater depth.

Detailed Summary of Insights

The analysis across collected datasets, visualizations, and statistical comparisons highlights critical insights about CPU performance measurement inaccuracies, interrelations between metrics, and system behaviors under various loads. Below is a detailed breakdown of the insights derived:

1. **Energy Efficiency: Operating System Differences**

- **Linux Efficiency**: Linux demonstrated significantly better energy efficiency compared to Windows, particularly in terms of energy per instruction (EPI). The tight clustering of Linux's EPI values reflects its optimized instruction scheduling and system management.

- **Windows Inefficiencies**:
    - Cache utilization was markedly less efficient on Windows, as indicated by significantly higher cache misses per Joule.
    - Greater variability in EPI suggests inconsistencies in workload handling, potentially caused by less effective power management or task scheduling policies.

Key Supporting Evidence:

- **Statistics**: Significant differences were noted in t-tests for EPI and cache metrics, with Linux outperforming Windows in both cases.

2. Anomalous Energy Consumption Trends

- **Counterintuitive Energy Patterns**:
    - Energy consumption peaked at ~30–40% CPU load but unexpectedly decreased at higher loads. This behavior contradicts typical expectations, where energy consumption should rise proportionally with load.
    - **Hypotheses**:
        - **CPU Throttling**: CPUs may reduce clock speeds at higher loads to prevent overheating.
        - **Measurement Inaccuracies**: PCM's data collection overhead or calibration issues could have introduced systematic errors.
- **Mitigation**:
    - Extended the duration of data collection to smooth out transient anomalies.

    - Focused on high-load scenarios to examine throttling-related patterns.

Key Supporting Evidence:

- **Observations**: Clear dips in energy consumption at high loads were consistently observed across multiple datasets.

3. Temperature vs. Frequency Insights

- **Observed Trends**:
    - Temperatures increased with CPU frequency but plateaued at high loads, indicating active thermal management (e.g., throttling or enhanced cooling mechanisms).
    - Flat-line temperature data suggested resolution limitations in sensors, especially for small temperature variations.
- **Challenges**:
    - Whole-number precision from sensors reduced the accuracy of thermal trend analysis.
    - Attempts to use hwmon for finer-grain temperature data showed promise but were unreliable due to lack of real-time updates.
- **Mitigation**:
    - Implemented interpolation methods to bridge gaps in temperature data.
    - Returned to sensors for consistent (if limited) results.

Key Supporting Evidence:

- **Findings**: Smooth trends at lower loads transitioned into plateaus, aligning with hypothesized throttling mechanisms.

4. Statistical Correlations and Relationships

- **Strong Correlations**:
    - CPU utilization strongly correlated with energy consumption and temperature, affirming expectations of proportional relationships under normal conditions.
    - However, higher utilization often produced diminishing returns on energy, hinting at throttling or energy inefficiency under stress.
- **Weak Correlations**:
    - Cache misses showed weak correlation with energy metrics, suggesting inefficiencies in cache handling or task scheduling on Windows.
    - Higher cache misses per Joule were consistently observed on Windows, likely contributing to its lower energy efficiency.

Key Supporting Evidence:

- **Analysis**: Heatmaps and scatter plots confirmed the relationships between key metrics and highlighted areas where systems diverged significantly.

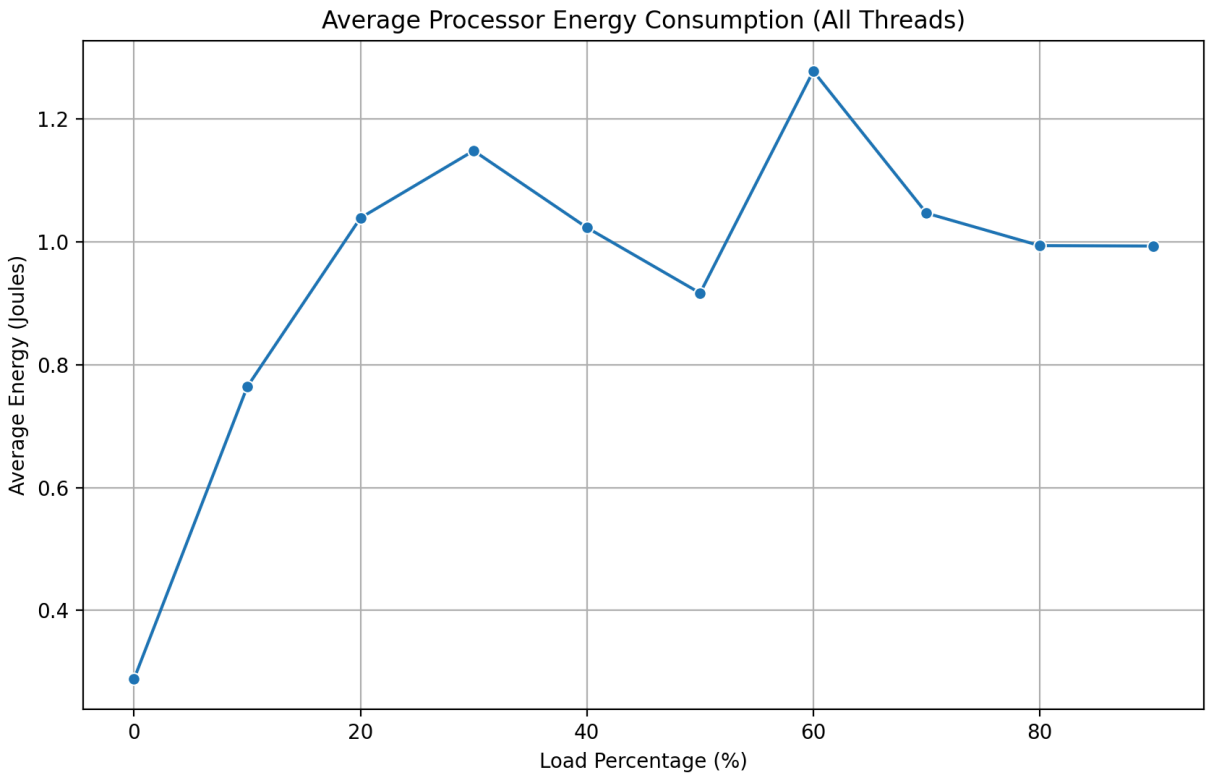
5. Persistent Data Inaccuracies

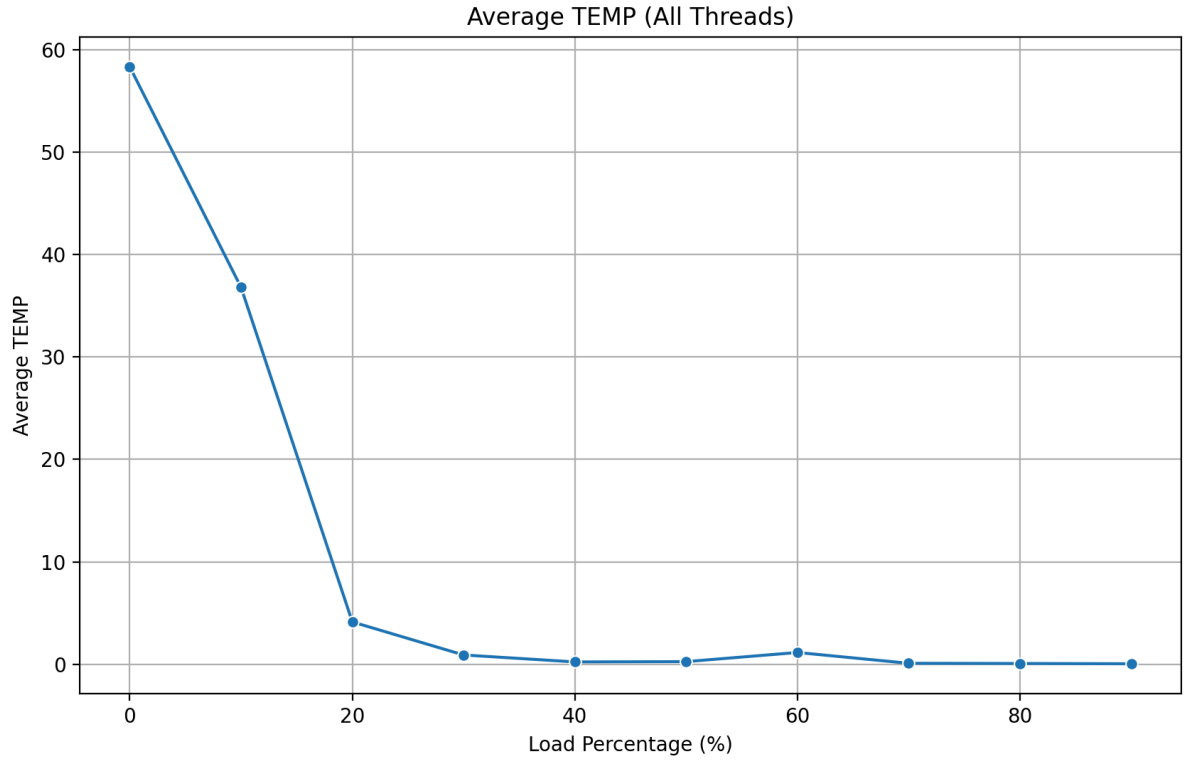
Despite extensive preprocessing and filtering, certain inaccuracies persisted, offering insights into the limitations of current tools:

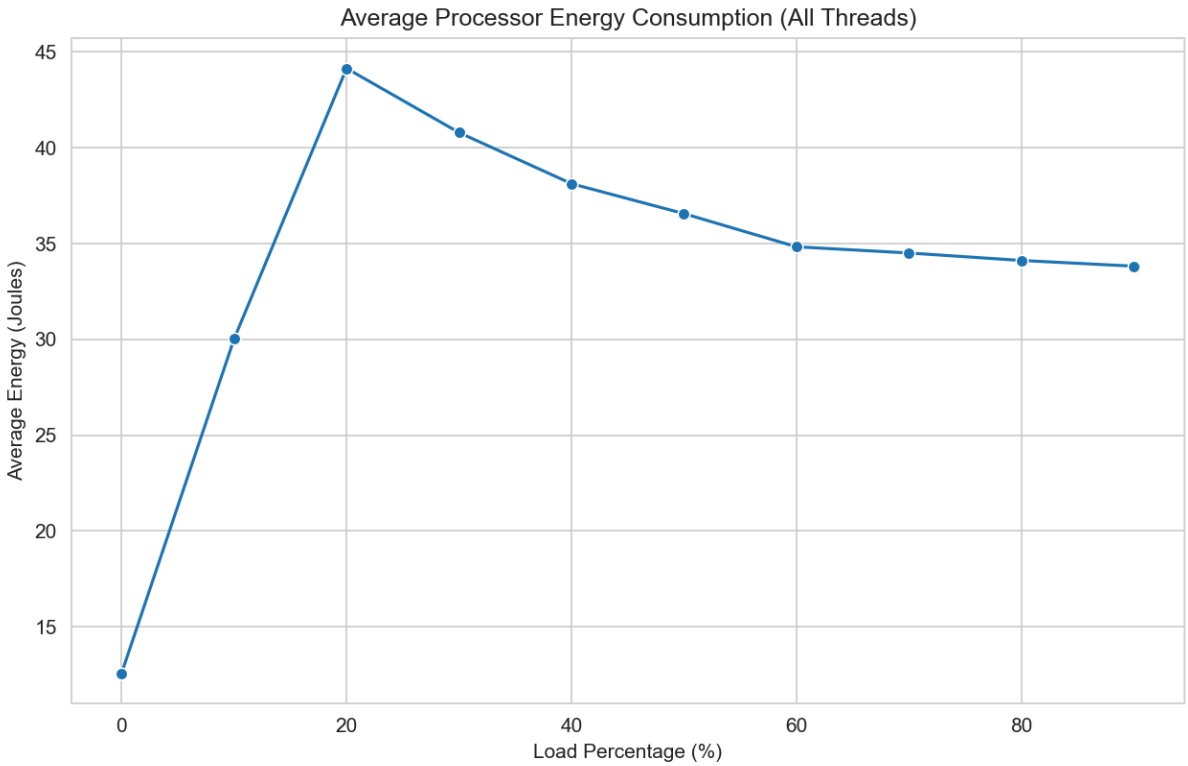
- **Negative Energy Values**:
    - PCM occasionally reported negative energy values, which were logically invalid. These were attributed to potential counter overflows or misalignments in tool calibration.
- **Over-Utilization**:
    - CPU utilization exceeded 100% in multi-threaded tasks, particularly on Windows, likely due to hyper-threading or scheduling artifacts.
- **Line Count Mismatches**:
    - Temperature and PCM data often had mismatched timestamps, requiring manual interpolation and alignment.

Key Supporting Evidence:

- **Images**:


Average Processor Energy Consumption (All Threads)
Average Energy (Joules)
1.2
1.0
0.8
0.6
0.4
0
20
40
60
80
Load Percentage (%)


Average TEMP (All Threads)
Average TEMP
Load Percentage (%)
0
10
20
30
40
50
60
0
20
40
60
80


Average Processor Energy Consumption (All Threads)
Average Energy (Joules)
Load Percentage (%)
15
20
25
30
35
40
45
0
20
40
60
80

- **Mitigations**:
    - Excluded anomalous data points during preprocessing.
    - Adjusted sampling intervals to synchronize outputs from different tools.

Final Reflections: This analysis underscores the inherent limitations of current CPU monitoring tools and the need for comprehensive validation. While Linux generally outperformed Windows in energy efficiency and cache utilization, significant inaccuracies persisted in both environments. Addressing these challenges requires a multifaceted approach, combining better tools, refined methodologies, and cross-validation with external benchmarks.

Summary and Conclusion

Summary

This project initially aimed to assess and highlight the efficiencies and inefficiencies of current operating systems by examining specific instructions that OSes pass to the CPU. However, the focus evolved into investigating the inaccuracies in CPU performance data collected from monitoring tools across Windows and Linux environments. The findings demonstrated a range of limitations in existing tools, revealed significant discrepancies in energy consumption patterns and efficiency metrics, and offered insights into the interaction between system behaviors and tool-induced inaccuracies.

Despite these challenges, I successfully analyzed the data to compare the energy efficiencies of Windows and Linux operating systems, as well as between two different CPU architectures within the same Linux version. Specifically, the study compared the i7-7700K and i7-13700K CPUs, uncovering notable differences in energy efficiency and CPU utilization behaviors. However, due to a lack of access to additional Windows 10 machines, the analysis was limited in its ability to thoroughly compare operating systems and to evaluate the same CPUs across different Windows versions.

The shift in research focus underscored significant issues within the data collection process, particularly concerning the reliability of monitoring tools like PCM. The project highlighted the necessity for more accurate measurement tools and methodologies to ensure the integrity of CPU performance data.

Key Conclusions

Energy Efficiency

- **Linux Superiority:** Linux consistently outperformed Windows in energy efficiency metrics, particularly in Energy per Instruction (EPI). This underscores the impact of operating system design on power management and task scheduling efficiency.
- **Windows Variability:** Windows exhibited greater variability in Cache Misses per Joule, indicating inefficiencies in cache utilization and higher instruction overhead.

Inaccuracies in Data Collection

- **Recurring Issues:** Negative energy values, CPU utilizations exceeding 100%, and line mismatches between PCM and temperature sensors were recurrent problems.
- **Tool-Specific Limitations:** These anomalies originated from tool-specific limitations, including sampling overhead, synchronization issues, and hardware counter inaccuracies.

Temperature and Throttling

- **Thermal Management:** Temperature data plateaued at higher loads, suggesting effective thermal throttling mechanisms, particularly on Linux systems. However, the lack of fine-grained temperature data obscured subtle variations that could provide deeper insights into thermal behaviors.

Behavior Under Load

- **Counterintuitive Patterns:** Energy consumption peaked around 30–40% CPU load and decreased at higher loads. This unexpected pattern suggested either CPU throttling or inaccuracies in PCM energy reporting, necessitating further investigation.

Correlations and Efficiency Metrics

- **Strong Relationships:** There were strong correlations between CPU utilization, energy consumption, and temperature.
- **Cache Inefficiencies:** Weak correlations between cache metrics and energy consumption highlighted areas needing improvement in resource management.

Broader Implications

- **Need for Reliable Tools:** The findings emphasize the necessity for reliable, high-precision tools and methodologies in CPU performance analysis.
- **Validation Importance:** The observed discrepancies underscore the importance of validating tool outputs with independent measurements to ensure data accuracy.

Recommendations for Future Studies

To address the limitations encountered in this study and to further advance the understanding of CPU performance and monitoring tool accuracy, the following comprehensive recommendations are proposed:

1. **Tool Enhancement and Validation**
    - **Cross-Validation with External Hardware:**
        - Integrate external power meters (e.g., Kill-A-Watt or advanced energy measurement hardware) to validate PCM energy readings, helping to identify systematic biases and ensure data reliability.
        - Use infrared thermography or external temperature sensors to verify onboard sensor data.
    - **Diversify Monitoring Tools:**
        - Explore advanced tools such as perf, eBPF, and Intel VTune for more kernel-level and microarchitectural insights.
        - Employ additional software profiling tools, like DynamoRIO, for instruction-level monitoring to cross-verify PCM outputs.

- **Calibrate Tools for Precision:**
  - Conduct controlled experiments with known energy and workload profiles to calibrate PCM and sensor outputs, especially for high-utilization or memory-intensive workloads.

2. **Improved Data Collection Techniques**
   - **Synchronization Strategies:**
     - Develop synchronized data collection frameworks that align timestamps across all tools in real time.
     - Utilize high-resolution timers and buffering mechanisms to ensure consistent line counts across data streams.
   - **Sampling Optimization:**
     - Evaluate the trade-offs between sampling frequency and monitoring overhead to balance precision with performance impact.
     - Implement dynamic sampling intervals, increasing frequency during high-load phases and reducing it during idle periods.
   - **Multi-Threaded and Multi-Core Scenarios:**
     - Extend the analysis to include a broader range of thread counts and hybrid-core CPUs. The increased accuracy of the i7-13700K's measurements suggests that modern CPUs provide a more dependable foundation for research.
     - Future studies could leverage this precision to perform more complex workload analysis, such as evaluating the relationship

between thread scheduling efficiency and energy consumption in multi-threaded applications.

- Investigate the impacts of newer instruction sets and enhanced monitoring capabilities on data fidelity to further validate findings and establish a baseline for benchmarking emerging CPUs.
- Conduct comparative studies that include other contemporary CPUs to determine whether the i7-13700K's superior measurement accuracy is unique to its architecture or reflective of broader industry trends in processor design and monitoring technology. This approach would ensure a comprehensive understanding of how modern CPUs behave under varied workloads and environmental conditions.

3. **Expanded Experimental Scenarios**
    - **Hybrid Architectures:**
        - Test on hybrid CPUs that combine performance and efficiency cores to study task allocation and its impact on energy efficiency.
        - Compare architectures, such as x86 and ARM, to evaluate cross-platform differences in energy consumption.
    - **Workload Diversity:**
        - Incorporate a variety of workloads, including:
            - **I/O-Intensive Tasks:** Analyze disk read/write operations for their impact on energy and performance.

        - **Memory-Intensive Tasks:** Examine cache and RAM utilization under high-stress scenarios.
        - **Real-World Applications:** Include benchmarks based on actual software, such as web browsers or machine learning workloads.
    - **Long-Term Analysis:**
        - Conduct prolonged experiments to identify wear-and-tear effects on CPUs and observe how performance changes over time.
4. **Enhanced Statistical and Analytical Approaches**
    - **Advanced Statistical Models:**
        - Utilize regression models to predict energy consumption based on workload, temperature, and CPU frequency.
        - Apply clustering algorithms to group similar workload behaviors and identify outliers.
    - **Machine Learning for Prediction:**
        - Train machine learning models to estimate energy consumption and identify anomalies based on collected metrics.
        - Employ techniques like ridge regression or neural networks to predict energy and performance trends from partial datasets.
    - **Expanded Visualization Techniques:**
        - Develop dynamic, interactive dashboards for real-time monitoring and analysis.

        - Create detailed 3D plots to visualize relationships between energy, temperature, and workload dimensions.

5. **Addressing Limitations in Temperature Sensing**
    - **Improved Sensor Hardware:**
        - Upgrade systems with high-precision thermal sensors capable of reporting data at finer resolutions (e.g., 0.001°C increments).
        - Validate temperature readings with external devices, such as thermal cameras.
    - **Custom Firmware Enhancements:**
        - Modify firmware to increase the resolution and real-time update rates of onboard sensors.
        - Explore using FPGA-based tools to enhance data collection precision.
    - **Tool Alternatives:**
        - Experiment with software-based approaches, such as lm-sensors with enhanced configurations, to derive more reliable temperature data.

Closing Reflections

The findings from this study not only highlight the limitations of existing CPU monitoring tools but also underscore the potential for significant improvements through validation, diversification, and methodological refinements. Addressing these challenges requires collaboration between software developers, hardware manufacturers, and researchers to develop tools and systems that provide reliable and actionable insights.

Bibliography

Raffin, G., & Trystram, D. (2024). Dissecting the Software-Based Measurement of CPU Energy Consumption: A Comparative Analysis. Retrieved from https://arxiv.org/abs/2401.15985

Myasnikov, V., Sartasov, S., Slesarev, I., & Gessen, P. (2020). Energy Consumption Measurement Frameworks for Android OS: A Systematic Literature Review. Proceedings of the 16th Central and Eastern European Software Engineering Conference (CEE-SECR 2020).

Tang, X., & Fu, Z. (2020). CPU–GPU Utilization Aware energy Efficient Scheduling Algorithm on Heterogeneous Computing Systems. IEEE Access, 8, 58948–58958. https://doi.org/10.1109/ACCESS.2020.2982956

Appendices

Source code can be found: https://github.com/nfantinodyer/CPUEnergyEstimation

All of my notes throughout this project are under the Notes directory in that same repository. Following the steps on the readme you can install pcm and run it on your pc as well. Initially I used Influx to try to push my data to which is what got me started with reformatting the csv files into the format influx needed, but I ended up still using that

CorrectData script throughout because I liked the consistency. For Linux I was able to automate the data collection using the mergingTempRun.sh since the amount of experiments I was running was tedious to run by hand. However for windows I wasn't given the opportunity to edit the amount of load on the cpu as well as some other things that I could do on linux so the data collection was minimal in comparison. The following files are the source code of what I wrote for this project to gather, clean, and analyze the data from the CPU's in windows and linux. If you were to test on your own pc make sure it's a linux machine or a windows 10 machine. You will need to change some of the code around (the headers in the correct data scripts) to match the output of your pcm data, as well as some of the directory locations.

```Readme.md
# CPU Performance Analysis: Addressing Inaccuracies in Monitoring Tools
## Overview
  This project investigates inaccuracies in CPU performance and energy consumption metrics collected using various tools, including Intel PCM, Linux sensors, and stress-testing utilities like stress-ng and HeavyLoad. The analysis compares results across Windows and Linux environments, identifying tool limitations, exploring mitigation strategies, and evaluating energy efficiency.

## Features
- Automated data collection for CPU performance metrics and temperature readings for Linux.
```

- Merged, cleaned, and preprocessed datasets ready for in-depth analysis.
- Visualizations, including scatter plots and heatmaps, to highlight trends and anomalies.
- Statistical analysis comparing Windows and Linux systems under varying loads and thread configurations.

Setup and Installation

Prerequisites

- Linux or Windows 10.
- It depends on the age of the motherboard, but if your motherboard supports secure boot, you will have to disable it for both windows and linux pcm to work.

Install and follow directions from:

https://github.com/intel/pcm

For Windows Intel Monitor I followed these steps

Installing PCM

https://github.com/intel/pcm

>git clone --recursive https://github.com/intel/pcm

>cd pcm

>git submodule update --init --recursive

>mkdir build

>cd build

Install cmake: https://cmake.org/download/

>cmake ..

>cmake --build .

>cmake --build . --parallel

>cmake --build . --config Release

https://github.com/intel/pcm/blob/master/doc/WINDOWS_HOWTO.md

To run PCM.exe

Disable Secure Boot first in BIOS

>bcdedit /set testsigning on

Only allow signed:

>bcdedit /set testsigning off

For perfmon:

in the pcm dir

>cmake --build build --target pcm-lib --config Release

>This will generate `PCM-Service.exe` in the same directory as other build outputs (e.g., `C:\Users\2013r\pcm\build\bin\Release`

Copy the following files into a `PCM` sub-directory in `C:\Program Files`:

- `PCM-Service.exe`
- `PCM-Service.exe.config`
- `pcm-lib.dll`

> cd "C:\Program Files\PCM"

> "PCM-Service.exe" -Install

Starting perfmon

> cd "C:\Program Files\PCM"

> net start pcmservice

- Open **Performance Monitor** (Perfmon) by typing `perfmon` in the Start menu and pressing Enter.
- In Perfmon, add new counters by clicking the green "+" icon.
- You should see new counters starting with `PCM*`, which are provided by the `PCM-Service.exe`.

Run the dataOut.ps1 file to get data and put it into a csv. It runs for 5 seconds and gets data very 250 milliseconds.

Just straight output

Turns out you can just do

>pcm.exe /csv 1 > output.csv

This will output to the csv every 1 second. You can also do every .1 seconds and that's what i have it on currently:

>pcm.exe /csv .1 > output.csv

For Linux I followed these steps

I'll be using Desbian edu 12.8.0 amd 64 netinst since it's primarily for Windows Intel CPUs. It'll be loaded onto the same drive as windows on the main testing PC so I can test on the same CPU.

Be sure to install

sudo apt-get update

sudo apt-get install stress-ng gcc g++ libacl1-dev libaio-dev libapparmor-dev libatomic1 libattr1-dev libbsd-dev libcap-dev libeigen3-dev libgbm-dev libcrypt-dev libglvnd-dev libipsec-mb-dev libjpeg-dev libjudy-dev libkeyutils-dev libkmod-dev libmd-dev libmpfr-dev libsctp-dev libxxhash-dev zlib1g-dev cmake

Go to the bin directory in the build one.

If you want to run it yourself manually you can do:

>sudo modprobe msr

> sudo ./pcm /csv .025 > ~/Desktop/LinuxOutput.csv 2>/dev/null

But if you want to have it automatically run expeiments you can run my mergingTempRun.sh file. It will output everything into the Data folder on your desktop:

You will need to edit the file to tell it where you installed pcm as well as where you need the output to go.

chmod +x mergingTempRun.sh

sudo ./mergingTempRun.sh

Some Notes

All of my notes throughout this project are under the Notes directory in that same repository. Following the steps on the readme you can install pcm and run it on your pc as well. Initially I used Influx to try to push my data to which is what got me started with reformatting the csv files into the format influx needed, but I ended up still using that CorrectData script throughout because I liked the consistency. For Linux I was able to automate the data collection using the mergingTempRun.sh since the amount of experiments I was running was tedious to run by hand. However for windows I wasn't given the opportunity to edit the amount of load on the cpu as well as some other things that I could do on linux so the data collection was minimal in comparison.

https://github.com/nfantinodyer/CPUEnergyEstimation

```

```allAnalyze.py
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import ttest_ind

# Set seaborn style
sns.set_style('whitegrid')

# Define data directories for Linux and Windows
base_dirs = {
    'Linux': 'Data/NewData/Linux/StressNGData',
    'Windows': 'Data/NewData/Windows'
}

# Define thread and load-specific files
linux_thread_dirs = {
    'All Threads': os.path.join(base_dirs['Linux'], 'Static/AllThreads'),
```
```

```
    '2 Threads': os.path.join(base_dirs['Linux'], 'Static/TwoThreads'),
    '4 Threads': os.path.join(base_dirs['Linux'], 'Static/FourThreads'),
    '6 Threads': os.path.join(base_dirs['Linux'], 'Static/SixThreads')
}
windows_thread_files = {
    '2 Threads': os.path.join(base_dirs['Windows'], 'Windows2threads.csv'),
    '4 Threads': os.path.join(base_dirs['Windows'], 'Windows4threads.csv'),
    '6 Threads': os.path.join(base_dirs['Windows'], 'Windows6threads.csv'),
    '8 Threads': os.path.join(base_dirs['Windows'], 'Windows8threads.csv')
}

# Load data functions
def load_linux_data(thread_dirs):
    all_data = []
    for thread_label, directory in thread_dirs.items():
        if not os.path.exists(directory):
            print(f"Directory not found: {directory}")
            continue
        for file in os.listdir(directory):
            file_path = os.path.join(directory, file)
            if os.path.isfile(file_path) and file.endswith('.csv'):
                print(f"Loading Linux file: {file_path}")
                df = pd.read_csv(file_path, skiprows=1)
```

```
            df['NumThreadsLabel'] = thread_label
            df['SourceFile'] = file
            df['OS'] = 'Linux'
            df.rename(columns={'Temperature': 'Temp'}, inplace=True)
            all_data.append(df)
    print("Linux data loaded.")
    return pd.concat(all_data, ignore_index=True) if all_data else pd.DataFrame()


def load_windows_data(thread_files):
    all_data = []
    for thread_label, file_path in thread_files.items():
        if not os.path.exists(file_path):
            print(f"File not found: {file_path}")
            continue
        print(f"Loading Windows file: {file_path}")
        df = pd.read_csv(file_path, skiprows=1)
        df['NumThreadsLabel'] = thread_label
        df['SourceFile'] = os.path.basename(file_path)
        df['OS'] = 'Windows'
        df.rename(columns={'TEMP': 'Temp'}, inplace=True)
        all_data.append(df)
    print("Windows data loaded.")
    return pd.concat(all_data, ignore_index=True) if all_data else pd.DataFrame()
```

```
# Load both datasets
linux_data = load_linux_data(linux_thread_dirs)
windows_data = load_windows_data(windows_thread_files)

print("Linux data sample:")
print(linux_data.head())
print("\nWindows data sample:")
print(windows_data.head())

# Clean and deduplicate data
linux_numeric_data = linux_data.select_dtypes(include=[np.number])
windows_numeric_data = windows_data.select_dtypes(include=[np.number])

# Remove duplicate columns
linux_numeric_data = linux_numeric_data.loc[:,
~linux_numeric_data.columns.duplicated()]
windows_numeric_data = windows_numeric_data.loc[:,
~windows_numeric_data.columns.duplicated()]

print("\nLinux numeric data columns:")
print(linux_numeric_data.columns.tolist())
print("\nWindows numeric data columns:")
```

```
print(windows_numeric_data.columns.tolist())


# Column categorization for heatmaps
categories = {
    "CPU and Energy Metrics": [
        "CPU Utilization", "Energy Consumption (Joules)", "Frequency", "Actual
Frequency", "Temp"
    ],
    "Cache and Memory Metrics": [
        "L3MISS", "L2MISS", "IPC", "EXEC", "INSTnom", "TIME(ticks)", "PhysIPC"
    ],
    "Utilization Metrics": [
        "CPU Utilization", "C0res%", "C3res%", "C6res%", "C7res%"
    ],
    "Performance Metrics": [
        "IPC", "EXEC", "INSTnom", "TIME(ticks)", "PhysIPC"
    ]
}


# Function for heatmaps
def plot_heatmap(data, title):
    if data.shape[1] < 2:
        print(f"Skipping heatmap for '{title}' due to insufficient data.")
```

```
        return
    print(f"\nGenerating heatmap for '{title}'")
    print(data.corr())
    plt.figure(figsize=(12, 10))
    sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
    plt.title(title)
    plt.show()

# Function for scatterplots
def scatter_plot(data, x, y, hue, title):
    if x not in data.columns or y not in data.columns:
        print(f"Skipping scatter plot for '{title}' due to missing columns.")
        return
    print(f"\nScatter plot data for '{title}':")
    print(data[[x, y, hue]].dropna().head())
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=data, x=x, y=y, hue=hue, palette='viridis', alpha=0.7)
    plt.title(title)
    plt.xlabel(x)
    plt.ylabel(y)
    plt.grid(True)
    plt.legend(title=hue)
    plt.show()
```

```
# Function for line plots
def line_plot(data, x, y, hue, title):
    if x not in data.columns or y not in data.columns:
        print(f"Skipping line plot for '{title}' due to missing columns.")
        return
    print(f"\nLine plot data for '{title}':")
    print(data[[x, y]].dropna().head())
    plt.figure(figsize=(10, 6))
    sns.lineplot(data=data, x=x, y=y, hue=hue, marker='o')
    plt.title(title)
    plt.xlabel(x)
    plt.ylabel(y)
    plt.grid(True)
    plt.show()


# Generate and plot heatmaps for each category
for category, cols in categories.items():
    print(f"\nProcessing category: {category}")
    linux_subset = linux_numeric_data[linux_numeric_data.columns.intersection(cols)]
    windows_subset =
windows_numeric_data[windows_numeric_data.columns.intersection(cols)]
```

```
    if not linux_subset.empty:
        plot_heatmap(linux_subset, f"Linux Heatmap: {category}")
    if not windows_subset.empty:
        plot_heatmap(windows_subset, f"Windows Heatmap: {category}")

# Scatterplots for Cache and Memory Metrics
scatter_plot(linux_data, "L3MISS", "IPC", "NumThreadsLabel", "Linux: IPC vs. L3MISS by Threads")
scatter_plot(windows_data, "L3MISS", "IPC", "NumThreadsLabel", "Windows: IPC vs. L3MISS by Threads")

# Scatterplots for Performance Metrics
scatter_plot(linux_data, "EXEC", "INSTnom", "NumThreadsLabel", "Linux: EXEC vs. INSTnom by Threads")
scatter_plot(windows_data, "EXEC", "INSTnom", "NumThreadsLabel", "Windows: EXEC vs. INSTnom by Threads")

# Additional correlations (cache vs. power utilization)
scatter_plot(linux_data, "L3MISS", "Proc Energy (Joules)", "NumThreadsLabel", "Linux: L3MISS vs. Proc Energy by Threads")
scatter_plot(windows_data, "L3MISS", "Proc Energy (Joules)", "NumThreadsLabel", "Windows: L3MISS vs. Proc Energy by Threads")
```

```
# Line plots for EXEC vs. INSTnom
line_plot(linux_numeric_data, "INSTnom", "EXEC", None, "Linux: EXEC vs. INSTnom")
line_plot(windows_numeric_data, "INSTnom", "EXEC", None, "Windows: EXEC vs.
INSTnom")

# Energy efficiency calculations and visualizations
def compute_energy_efficiency(data):
    if 'INST' in data.columns and 'TIME(ticks)' in data.columns and 'Proc Energy (Joules)'
in data.columns:
        data['Energy per Instruction'] = data['Proc Energy (Joules)'] / data['INST']
        data['Energy per Cycle'] = data['Proc Energy (Joules)'] / data['TIME(ticks)']
        data['Cache Miss per Joule'] = data['L3MISS'] / data['Proc Energy (Joules)']
    return data

linux_data = compute_energy_efficiency(linux_data)
windows_data = compute_energy_efficiency(windows_data)

# Summarize energy efficiency
def summarize_efficiency(data, os_name):
    print(f"\n{os_name} Energy Efficiency Summary:")
    summary = data[['Energy per Instruction', 'Energy per Cycle', 'Cache Miss per
Joule']].describe().T
    print(summary)
```

```
    return summary

linux_summary = summarize_efficiency(linux_data, "Linux")
windows_summary = summarize_efficiency(windows_data, "Windows")

# Scatterplots for energy efficiency
scatter_plot(linux_data, "NumThreadsLabel", "Energy per Instruction",
"NumThreadsLabel", "Linux: Energy per Instruction vs. Threads")
scatter_plot(windows_data, "NumThreadsLabel", "Energy per Instruction",
"NumThreadsLabel", "Windows: Energy per Instruction vs. Threads")
scatter_plot(linux_data, "NumThreadsLabel", "Energy per Cycle", "NumThreadsLabel",
"Linux: Energy per Cycle vs. Threads")
scatter_plot(windows_data, "NumThreadsLabel", "Energy per Cycle",
"NumThreadsLabel", "Windows: Energy per Cycle vs. Threads")
scatter_plot(linux_data, "NumThreadsLabel", "Cache Miss per Joule",
"NumThreadsLabel", "Linux: Cache Miss per Joule vs. Threads")
scatter_plot(windows_data, "NumThreadsLabel", "Cache Miss per Joule",
"NumThreadsLabel", "Windows: Cache Miss per Joule vs. Threads")

# Compare energy efficiency metrics across Linux and Windows
def compare_efficiency(linux_data, windows_data, metric, metric_name):
    linux_values = linux_data[metric].dropna()
    windows_values = windows_data[metric].dropna()
```

```
    print(f"\nComparing {metric_name} between Linux and Windows:")
    t_stat, p_value = ttest_ind(linux_values, windows_values, equal_var=False)
    print(f"T-statistic: {t_stat}, P-value: {p_value}")

    plt.figure(figsize=(10, 6))
    sns.boxplot(data=[linux_values, windows_values], notch=True)
    plt.xticks([0, 1], ["Linux", "Windows"])
    plt.title(f"Comparison of {metric_name} between Linux and Windows")
    plt.ylabel(metric_name)
    plt.show()

# Compare energy efficiency metrics
compare_efficiency(linux_data, windows_data, "Energy per Instruction", "Energy per Instruction")
compare_efficiency(linux_data, windows_data, "Energy per Cycle", "Energy per Cycle")
compare_efficiency(linux_data, windows_data, "Cache Miss per Joule", "Cache Miss per Joule")

print("Full analysis complete.")
```

```CorrectDataLinux.ps1
```

```
$inputDirectory = "Data\Linux\StressNGData"
$outputDirectory = "Data\NewData\Linux\StressNGData"
if (!(Test-Path $outputDirectory)) {
    New-Item -Path $outputDirectory -ItemType Directory -Force | Out-Null
}

# Resolve to full absolute paths
$inputDirectory = (Get-Item -Path $inputDirectory).FullName
$outputDirectory = (Get-Item -Path $outputDirectory).FullName

# Measurement name
$measurement = "LinuxPCM"

# Define headers and datatype rows for i7-7700K
$headerRow_7700K = $measurement +
",DateTime,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2
MPI,READ,WRITE,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,
C0res%,C1res%,C3res%,C6res%,C7res%,C0res%,C2res%,C3res%,C6res%,C7res%,
C8res%,C9res%,C10res%,Proc Energy (Joules),Power Plane 0 Energy (Joules),Power
Plane 1 Energy
(Joules),EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI
,READ,WRITE,IO,IA,GT,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,I
NSTnom%,C0res%,C1res%,C3res%,C6res%,C7res%,C0res%,C2res%,C3res%,C6res
```

%,C7res%,C8res%,C9res%,C10res%,SKT0,SKT0,SKT0,EXEC,IPC,FREQ,AFREQ,CF
REQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7
res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,I
PC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res
%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn
om,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI
,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysI
PC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MIS
S,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,AC
YC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,C
FREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C
7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,
IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res
%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn
om,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI
,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysI
PC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MIS
S,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,AC
YC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,,Temperature"

$datatypeRow_7700K = "#datatype
measurement,dateTime:2006-01-02T15:04:05.000Z,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do

uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,,double,double,double,double,double,double,d
ouble,double,double,double,double"

Define headers and datatype rows for i7-13700K

$headerRow_13700K = $measurement +

",DateTime,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2

MPI,READ,WRITE,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,

C0res%,C1res%,C3res%,C6res%,C7res%,C0res%,C2res%,C4res%,C6res%,Proc

Energy (Joules),Power Plane 0 Energy (Joules),Power Plane 1 Energy

(Joules),SYSTEM Energy

(Joules),EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI

,READ,WRITE,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom

%,C0res%,C1res%,C3res%,C6res%,C7res%,C0res%,C2res%,C4res%,C6res%,SKT0,

SKT0,SKT0,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2

MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC

,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L

3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC

,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFR

EQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7r

es%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,I

PC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res

%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn

om,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI

,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysI

PC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MIS

S,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,AC

YC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,C

FREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C
7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,
IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res
%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn
om,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI
,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),Physl
PC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MIS
S,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,AC
YC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,C
FREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C
7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,
IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res
%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn
om,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI
,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),Physl
PC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MIS
S,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,AC
YC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,C
FREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C
7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,
IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res
%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn
om,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI

,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysI
PC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MIS
S,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,AC
YC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,C
FREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C
7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,
IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res
%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn
om,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI
,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysI
PC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MIS
S,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,AC
YC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,C
FREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C
7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,
IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L2HIT,L3MPI,L2MPI,C0res%,C1res
%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTn
om,INSTnom%,,Temperature"

$datatypeRow_13700K = "#datatype
measurement,dateTime:2006-01-02T15:04:05.000Z,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d

ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou

ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,

```
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double"

# Get all input files
$inputFiles = Get-ChildItem -Path $inputDirectory -Filter "*.csv" -Recurse

foreach ($inputFile in $inputFiles) {
    $use13700K = $inputFile.FullName -like "*i713700K*"

    if ($use13700K) {
        $headerRow = $headerRow_13700K
        $datatypeRow = $datatypeRow_13700K
    } else {
        $headerRow = $headerRow_7700K
        $datatypeRow = $datatypeRow_7700K
    }

    # Ensure consistent path formats
    $inputFileFullName = $inputFile.FullName -replace '\\', '/'
```

```
$inputDirNormalized = $inputDirectory -replace '\\', '/'

$relativePath = $inputFileFullName.Substring($inputDirNormalized.Length)
$relativePath = $relativePath.TrimStart('/', '\')

$outputFilePath = Join-Path $outputDirectory $relativePath

$outputFileDirectory = Split-Path $outputFilePath -Parent
if (!(Test-Path $outputFileDirectory)) {
    New-Item -Path $outputFileDirectory -ItemType Directory -Force | Out-Null
}

$lines = Get-Content -Path $inputFile.FullName

$adjustedLines = @()
$adjustedLines += $datatypeRow
$adjustedLines += $headerRow

# Adjust the loop to exclude the last row
foreach ($line in $lines[2..($lines.Count - 2)]) {
    $adjustedLine = $line -replace '(^\d{4}-\d{2}-\d{2}),(\d{2}:\d{2}:\d{2}\.\d+)', '$1T$2Z'
    $adjustedLine = $measurement + "," + $adjustedLine
    $adjustedLines += $adjustedLine
```

```
    }

    $adjustedLines | Set-Content -Path $outputFilePath
}

```

```CorrectDataWindows.ps1
# Set directories
$inputDirectory = "Data\Windows"
$outputDirectory = "Data\NewData\Windows"

# Create output directory if it doesn't exist
if (!(Test-Path $outputDirectory)) {
    New-Item -Path $outputDirectory -ItemType Directory -Force | Out-Null
}

# Resolve to full absolute paths
$inputDirectory = (Get-Item -Path $inputDirectory).FullName
$outputDirectory = (Get-Item -Path $outputDirectory).FullName

# Define headers and datatype rows
```

```
$measurement = "WindowsPCM"
$headerRow = $measurement +
",DateTime,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L3MPI,L2MPI,RE
AD,WRITE,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,C0res%,
C1res%,C3res%,C6res%,C7res%,C0res%,C2res%,C3res%,C6res%,C7res%,C8res%,
C9res%,C10res%,Proc Energy (Joules),Power Plane 0 Energy (Joules),Power Plane 1
Energy
(Joules),EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L3MPI,L2MPI,REA
D,WRITE,IO,IA,GT,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTn
om%,C0res%,C1res%,C3res%,C6res%,C7res%,C0res%,C2res%,C3res%,C6res%,C7r
es%,C8res%,C9res%,C10res%,SKT0,SKT0,SKT0,EXEC,IPC,FREQ,AFREQ,CFREQ,L
3MISS,L2MISS,L3HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,I
NST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AF
REQ,CFREQ,L3MISS,L2MISS,L3HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,
C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXE
C,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L3MPI,L2MPI,C0res%,C1res%,C
3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,I
NSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L3MPI,L2MPI,C0
res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIP
C%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L3
MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),P
hysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CFREQ,L3MISS,L2
MISS,L3HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,TEMP,INST,ACY
```

C,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,FREQ,AFREQ,CF
REQ,L3MISS,L2MISS,L3HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,C6res%,C7res%,
TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTnom%,EXEC,IPC,F
REQ,AFREQ,CFREQ,L3MISS,L2MISS,L3HIT,L3MPI,L2MPI,C0res%,C1res%,C3res%,
C6res%,C7res%,TEMP,INST,ACYC,TIME(ticks),PhysIPC,PhysIPC%,INSTnom,INSTno
m%,"
$datatypeRow = "#datatype
measurement,dateTime:2006-01-02T15:04:05.000Z,double,double,double,,double,doub
le,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl

```
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,double,double,double,double,double,double,double,do
uble,double,double,double,double,double,double,double,double,double,double,double,d
ouble,double,double,double,double,double,double,double,double,double,double,double,
double,double,double,double,double,double,double,double,double,double,double,doubl
e,double,double,double,double,double,double,double,double,double,double,double,dou
ble,double,double,double,double,"
# Get all CSV files in the input directory
$inputFiles = Get-ChildItem -Path $inputDirectory -Filter "*.csv" -Recurse

foreach ($inputFile in $inputFiles) {
    # Ensure consistent path formats
    $inputFileFullName = $inputFile.FullName -replace '\\', '/'
    $inputDirNormalized = $inputDirectory -replace '\\', '/'

    # Compute relative and output paths
    $relativePath = $inputFileFullName.Substring($inputDirNormalized.Length)
    $relativePath = $relativePath.TrimStart('/', '\')
    $outputFilePath = Join-Path $outputDirectory $relativePath

    # Ensure the output directory exists
    $outputFileDirectory = Split-Path $outputFilePath -Parent
    if (!(Test-Path $outputFileDirectory)) {
```

```
        New-Item -Path $outputFileDirectory -ItemType Directory -Force | Out-Null
    }

    # Read the input file
    $lines = Get-Content -Path $inputFile.FullName

    # Initialize adjusted lines
    $adjustedLines = @()
    $adjustedLines += $datatypeRow
    $adjustedLines += $headerRow

    # Adjust lines (skip the first row, add measurement column, and reformat timestamp)
    foreach ($line in $lines[2..($lines.Count - 2)]) {
        $adjustedLine = $line -replace '(^\d{4}-\d{2}-\d{2}),(\d{2}:\d{2}:\d{2}\.\d+)', '$1T$2Z'
        $adjustedLine = $measurement + "," + $adjustedLine
        $adjustedLines += $adjustedLine
    }

    # Write adjusted lines to the output file
    $adjustedLines | Set-Content -Path $outputFilePath
}
```

```LinuxAnalyze.py
```

```
import os
import re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from scipy.stats import ttest_ind

#Suppress SettingWithCopyWarning
pd.options.mode.chained_assignment = None

sns.set_style('whitegrid')

base_dirs = {
    'i77700K': 'Data/NewData/Linux/StressNGData',
    'i713700K': 'Data/NewData/Linux/StressNGData/i713700K'
}

directories = {
    'i77700K': {
```

```
        'All Threads': os.path.join(base_dirs['i77700K'], 'Static', 'AllThreads'),
        '2 Threads': os.path.join(base_dirs['i77700K'], 'Static', 'TwoThreads'),
        '4 Threads': os.path.join(base_dirs['i77700K'], 'Static', 'FourThreads'),
        '6 Threads': os.path.join(base_dirs['i77700K'], 'Static', 'SixThreads'),
    },
    'i713700K': {
        'All Threads': os.path.join(base_dirs['i713700K'], 'Static', 'AllThreads'),
        '2 Threads': os.path.join(base_dirs['i713700K'], 'Static', 'TwoThreads'),
        '4 Threads': os.path.join(base_dirs['i713700K'], 'Static', 'FourThreads'),
        '6 Threads': os.path.join(base_dirs['i713700K'], 'Static', 'SixThreads'),
    }
}

all_loads = list(range(0, 100, 10))
partial_loads = list(range(0, 100, 10))

def create_file_paths(directory, loads, suffix):
    return [os.path.join(directory, f'Linux{load}{suffix}.csv') for load in loads]

def load_data(files, num_threads_label, cpu_label):
    data_frames = []
    for file_path in files:
        if os.path.exists(file_path):
```

```
            print(f"Loading file: {file_path}")
            df = pd.read_csv(file_path, skiprows=1, header=0)
            df['DateTime'] = pd.to_datetime(df['DateTime'], errors='coerce')
            df.dropna(subset=['DateTime'], inplace=True)

            filename = os.path.basename(file_path)
            load_match = re.search(r'(\d+)', filename)
            load_percent = int(load_match.group(1)) if load_match else None

            df['LoadPercent'] = load_percent
            df['NumThreadsLabel'] = num_threads_label
            df['CPU'] = cpu_label
            df['SourceFile'] = filename

            data_frames.append(df)
        else:
            print(f"File not found: {file_path}")
    return data_frames

# Function to perform t-tests
def perform_t_tests(data, metric, thread_comparisons, cpu_label):
    results = []
    cpu_data = data[data['CPU'] == cpu_label]
```

```
    for group1, group2 in thread_comparisons:
        data1 = cpu_data[cpu_data['NumThreads'] == group1][metric]
        data2 = cpu_data[cpu_data['NumThreads'] == group2][metric]

        if len(data1) > 0 and len(data2) > 0:
            t_stat, p_value = ttest_ind(data1, data2, equal_var=False)
        else:
            t_stat, p_value = np.nan, np.nan

        results.append({
            'Group 1': group1,
            'Group 2': group2,
            't-statistic': t_stat,
            'p-value': p_value
        })

    return pd.DataFrame(results)

# Initialize data frames list
data_frames = []

# Iterate over each directory and load data
for cpu_label, cpu_dirs in directories.items():
```

```
    for num_threads_label, dir_path in cpu_dirs.items():
        if 'All' in num_threads_label:
            files = create_file_paths(dir_path, all_loads, 'Static')
        else:
            threads_num = int(num_threads_label.split()[0])
            files = create_file_paths(dir_path, partial_loads, f'Static{threads_num}threads')
        data_frames.extend(load_data(files, num_threads_label, cpu_label))

# Combine all data
all_data = pd.concat(data_frames, ignore_index=True)

# Define columns of interest
columnsOfInterest = [
    'DateTime', 'CPU', 'LoadPercent', 'NumThreadsLabel', 'SourceFile',
    'Proc Energy (Joules)', 'FREQ', 'AFREQ', 'Temperature', 'C0res%', 'C1res%',
'C3res%',
    'C6res%', 'C7res%', 'READ', 'WRITE'
]

# Function to get matching columns
def getMatchingColumns(columnsList, dataColumns):
    matchingColumns = {}
    for col in columnsList:
```

```
        matches = [c for c in dataColumns if c.startswith(col)]
        if matches:
            # For 'Temperature', pick the exact match or the first match
            if col == 'Temperature':
                temp_matches = [c for c in matches if c == 'Temperature']
                matchingColumns[col] = temp_matches[0] if temp_matches else matches[0]
            else:
                matchingColumns[col] = matches[0]
        else:
            print(f"Column '{col}' not found in data.")
    return matchingColumns

# Get matching columns
matchingColumns = getMatchingColumns(columnsOfInterest, all_data.columns)

# Filter and rename columns
all_data_filtered = all_data[list(matchingColumns.values())].copy()
all_data_filtered.columns = list(matchingColumns.keys())

# Data cleaning
# Replace zero temperatures with NaN
all_data_filtered['Temperature'] = all_data_filtered['Temperature'].replace(0, np.nan)
```

```
all_data_filtered['Temperature'] = pd.to_numeric(all_data_filtered['Temperature'],
errors='coerce')
all_data_filtered['Temperature'] =
all_data_filtered['Temperature'].interpolate(method='linear')
all_data_filtered.dropna(subset=['Temperature'], inplace=True)

# Map 'NumThreadsLabel' to number of threads
num_threads_mapping = {
    'All Threads': 8,
    '2 Threads': 2,
    '4 Threads': 4,
    '6 Threads': 6
}
all_data_filtered['NumThreads'] =
all_data_filtered['NumThreadsLabel'].map(num_threads_mapping)

# Rename 'C0res%' to 'CPU Utilization'
if 'C0res%' in all_data_filtered.columns:
    all_data_filtered.rename(columns={'C0res%': 'CPU Utilization'}, inplace=True)

all_data_filtered['Energy_per_Load'] = all_data_filtered['Proc Energy (Joules)'] /
all_data_filtered['LoadPercent']
```

```
all_data_filtered['Energy_per_Thread'] = all_data_filtered['Proc Energy (Joules)'] /
all_data_filtered['NumThreads']

all_data_filtered['Combined_Efficiency'] = (all_data_filtered['LoadPercent'] *
all_data_filtered['NumThreads']) / all_data_filtered['Proc Energy (Joules)']

# Create a separate DataFrame excluding LoadPercent = 0 for EPUL
epul_data = all_data_filtered[all_data_filtered['LoadPercent'] > 0].copy()
# Recalculate Energy_per_Load to ensure no division by zero
epul_data['Energy_per_Load'] = epul_data['Proc Energy (Joules)'] /
epul_data['LoadPercent']
# Update cpu_efficiency using epul_data
cpu_efficiency_epul = epul_data.groupby(['CPU', 'NumThreads', 'LoadPercent']).agg(
    AvgEnergy=('Proc Energy (Joules)', 'mean'),
    StdEnergy=('Proc Energy (Joules)', 'std'),
    AvgUtilization=('CPU Utilization', 'mean'),
    AvgTemperature=('Temperature', 'mean'),
    AvgEPUL=('Energy_per_Load', 'mean'),
    StdEPUL=('Energy_per_Load', 'std'),
    AvgEPT=('Energy_per_Thread', 'mean'),
    StdEPT=('Energy_per_Thread', 'std'),
    AvgCME=('Combined_Efficiency', 'mean'),
    StdCME=('Combined_Efficiency', 'std')
```

```
).reset_index()

# Group data by CPU, LoadPercent, and NumThreads for overall metrics
cpu_efficiency = all_data_filtered.groupby(['CPU', 'NumThreads', 'LoadPercent']).agg(
    AvgEnergy=('Proc Energy (Joules)', 'mean'),
    StdEnergy=('Proc Energy (Joules)', 'std'),
    AvgUtilization=('CPU Utilization', 'mean'),
    AvgTemperature=('Temperature', 'mean'),
    AvgEPUL=('Energy_per_Load', 'mean'),
    StdEPUL=('Energy_per_Load', 'std'),
    AvgEPT=('Energy_per_Thread', 'mean'),
    StdEPT=('Energy_per_Thread', 'std'),
    AvgCME=('Combined_Efficiency', 'mean'),
    StdCME=('Combined_Efficiency', 'std')
).reset_index()

# Visualization: Energy per Load Unit by Thread Count and CPU
plt.figure(figsize=(12, 6))
sns.lineplot(data=cpu_efficiency_epul, x='LoadPercent', y='AvgEPUL',
hue='NumThreads', style='CPU', marker='o')
plt.title('Energy per Load Unit (EPUL) by Thread Count and CPU')
plt.xlabel('Load Percentage (%)')
plt.ylabel('Average EPUL (Joules/%)')
```

```
plt.legend(title='Thread Count and CPU')
plt.grid(True)
plt.show()

# Visualization: Energy per Thread by Thread Count and CPU
plt.figure(figsize=(12, 6))
sns.lineplot(data=cpu_efficiency_epul, x='LoadPercent', y='AvgEPT',
hue='NumThreads', style='CPU', marker='o')
plt.title('Energy per Thread (EPT) by Thread Count and CPU')
plt.xlabel('Load Percentage (%)')
plt.ylabel('Average EPT (Joules/Thread)')
plt.legend(title='Thread Count and CPU')
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(data=cpu_efficiency_epul, x='LoadPercent', y='AvgCME',
hue='NumThreads', style='CPU', marker='o')
plt.title('Combined Efficiency Metric (CME) by Thread Count and CPU')
plt.xlabel('Load Percentage (%)')
plt.ylabel('Average CME ((% Load * Threads)/Joules)')
plt.legend(title='Thread Count and CPU')
plt.grid(True)
```

```
plt.show()

# Regression Analysis
regression_data = all_data_filtered[['Proc Energy (Joules)', 'LoadPercent',
'NumThreads', 'FREQ', 'Temperature', 'CPU Utilization', 'CPU']].dropna()

X = regression_data[['LoadPercent', 'NumThreads', 'FREQ', 'Temperature', 'CPU
Utilization']]
y = regression_data['Proc Energy (Joules)']

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit polynomial regression model
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X_scaled)

model = LinearRegression()
model.fit(X_poly, y)

# Predictions and metrics
y_pred = model.predict(X_poly)
```

```
mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y, y_pred)
print(f"Polynomial Regression - RMSE: {rmse:.4f}")
print(f"Polynomial Regression - R-squared: {r2:.4f}")

# Actual vs Predicted
plt.figure(figsize=(8, 6))
plt.scatter(y, y_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.title('Actual vs Predicted Energy Consumption')
plt.xlabel('Actual Energy (Joules)')
plt.ylabel('Predicted Energy (Joules)')
plt.grid(True)
plt.show()

# Residual Plot
residuals = y - y_pred
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.title('Residuals')
plt.xlabel('Predicted Energy (Joules)')
```

```
plt.ylabel('Residuals (Joules)')
plt.grid(True)
plt.show()

# CPU-specific scatterplots
plt.figure(figsize=(8, 6))
sns.scatterplot(data=regression_data, x='CPU Utilization', y='Proc Energy (Joules)',
hue='CPU')
plt.title('Energy Consumption vs CPU Utilization by CPU')
plt.xlabel('CPU Utilization (%)')
plt.ylabel('Energy Consumption (Joules)')
plt.legend(title='CPU')
plt.grid(True)
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(data=regression_data, x='FREQ', y='Proc Energy (Joules)', hue='CPU')
plt.title('Energy Consumption vs Frequency by CPU')
plt.xlabel('Frequency (MHz)')
plt.ylabel('Energy Consumption (Joules)')
plt.legend(title='CPU')
plt.grid(True)
plt.show()
```

```
# Function to generate heatmap
def generate_heatmap(data, title):
    correlation_data = data[['LoadPercent', 'Proc Energy (Joules)', 'FREQ', 'AFREQ',
'Temperature',
                             'CPU Utilization', 'C1res%', 'C3res%',
                             'C6res%', 'C7res%', 'READ', 'WRITE']].corr()
    print(f"\nCorrelation Data for {title}:")
    print(correlation_data)
    plt.figure(figsize=(12, 10))
    sns.heatmap(correlation_data, annot=True, cmap="coolwarm", fmt=".2f")
    plt.title(title)
    plt.show()

# Filter data for each CPU and generate heatmaps
data_i7_7700K = all_data_filtered[all_data_filtered['CPU'] == 'i77700K']
data_i7_13700K = all_data_filtered[all_data_filtered['CPU'] == 'i713700K']

generate_heatmap(data_i7_7700K, "i7-7700K Heatmap")
generate_heatmap(data_i7_13700K, "i7-13700K Heatmap")

print("\nKey Metrics for i7-7700K:")
print(data_i7_7700K.describe())
```

```
print("\nKey Metrics for i7-13700K:")
print(data_i7_13700K.describe())

print("\nSummary of Key Metrics:")
print(all_data_filtered[['CPU', 'LoadPercent', 'Proc Energy (Joules)', 'Temperature', 'CPU
Utilization']].describe())

print("\nAggregated Metrics by CPU and Load Percentage:")
print(cpu_efficiency_epul)

print("\nCorrelation Matrix of Key Metrics:")
correlation_matrix = all_data_filtered[['Proc Energy (Joules)', 'LoadPercent',
'Temperature', 'CPU Utilization']].corr()
print(correlation_matrix)

print("\nOutlier Detection - Rows with Extreme Energy Consumption:")
outliers = all_data_filtered[all_data_filtered['Proc Energy (Joules)'] >
all_data_filtered['Proc Energy (Joules)'].quantile(0.99)]
print(outliers)

print("\nOutlier Detection - Rows with Extreme Energy per Load:")
```

```
outliers_epul = epul_data[epul_data['Energy_per_Load'] >
epul_data['Energy_per_Load'].quantile(0.99)]
print(outliers_epul)

print("\nOutlier Detection - Rows with Extreme Energy per Thread:")
outliers_ept = all_data_filtered[all_data_filtered['Energy_per_Thread'] >
all_data_filtered['Energy_per_Thread'].quantile(0.99)]
print(outliers_ept)

print("\nThread-Level Energy and Temperature Averages:")
thread_averages = all_data_filtered.groupby(['NumThreadsLabel', 'LoadPercent']).agg({
    'Proc Energy (Joules)': 'mean',
    'Temperature': 'mean',
    'CPU Utilization': 'mean',
    'Energy_per_Load': 'mean',
    'Energy_per_Thread': 'mean'
}).reset_index()
print(thread_averages)

# Define thread comparisons
thread_comparisons = [
    (2, 4),
    (4, 6),
```

```
    (2, 6)
]

# Function to perform t-tests for energy efficiency metrics
def perform_t_tests_efficiency(data, metric, thread_comparisons, cpu_label):
    results = []
    cpu_data = data[data['CPU'] == cpu_label]
    for group1, group2 in thread_comparisons:
        data1 = cpu_data[cpu_data['NumThreads'] == group1][metric]
        data2 = cpu_data[cpu_data['NumThreads'] == group2][metric]

        if len(data1) > 0 and len(data2) > 0:
            t_stat, p_value = ttest_ind(data1, data2, equal_var=False)
        else:
            t_stat, p_value = np.nan, np.nan

        results.append({
            'Group 1': group1,
            'Group 2': group2,
            't-statistic': t_stat,
            'p-value': p_value
        })
```

```
    return pd.DataFrame(results)

# Perform t-tests for Energy Consumption
energy_test_results = perform_t_tests(
    data=all_data_filtered,
    metric='Proc Energy (Joules)',
    thread_comparisons=thread_comparisons,
    cpu_label='i713700K'
)

print("\nT-Test Results for Energy Consumption (i7-13700K):")
print(energy_test_results)

# Perform t-tests for Energy per Load Unit (EPUL)
energy_efficiency_tests_epul = perform_t_tests_efficiency(
    data=epul_data,
    metric='Energy_per_Load',
    thread_comparisons=thread_comparisons,
    cpu_label='i713700K'
)

print("\nT-Test Results for Energy per Load Unit (EPUL) (i7-13700K):")
print(energy_efficiency_tests_epul)
```

```
# Perform t-tests for Energy per Thread (EPT)
energy_efficiency_tests_ept = perform_t_tests_efficiency(
    data=all_data_filtered,
    metric='Energy_per_Thread',
    thread_comparisons=thread_comparisons,
    cpu_label='i713700K'
)

print("\nT-Test Results for Energy per Thread (EPT) (i7-13700K):")
print(energy_efficiency_tests_ept)

# Perform t-tests for Temperature
temperature_test_results = perform_t_tests(
    data=all_data_filtered,
    metric='Temperature',
    thread_comparisons=thread_comparisons,
    cpu_label='i713700K'
)

print("\nT-Test Results for Temperature (i7-13700K):")
print(temperature_test_results)
```

```
# Scatterplot: Energy vs. Threads (i7-13700K)
plt.figure(figsize=(12, 6))
sns.scatterplot(data=all_data_filtered[all_data_filtered['CPU'] == 'i713700K'],
                x='NumThreadsLabel',
                y='Proc Energy (Joules)',
                hue='LoadPercent',
                palette='viridis')
plt.title('Energy Consumption by Threads (i7-13700K)')
plt.xlabel('Thread Group')
plt.ylabel('Energy Consumption (Joules)')
plt.legend(title='Load Percentage')
plt.grid(True)
plt.show()

# Scatterplot: Temperature vs. Threads (i7-13700K)
plt.figure(figsize=(12, 6))
sns.scatterplot(data=all_data_filtered[all_data_filtered['CPU'] == 'i713700K'],
                x='NumThreadsLabel',
                y='Temperature',
                hue='LoadPercent',
                palette='coolwarm')
plt.title('Temperature by Threads (i7-13700K)')
plt.xlabel('Thread Group')
```

```
plt.ylabel('Temperature (°C)')

plt.legend(title='Load Percentage')

plt.grid(True)

plt.show()
```

```mergingTempRun.sh
#!/bin/bash

# Ensure you run this script with sudo as pcm requires root privileges

# Directories

PCM_DIR="/home/rob/Desktop/pcm/build/bin"

OUTPUT_DIR="/home/rob/Desktop/Data"

SAMPLING_INTERVAL=1  # Same as PCM sampling interval

TOTAL_DURATION=30    # 30 seconds

# Ensure the msr module is loaded

sudo modprobe msr

# Function to collect temperature data and output to stdout

collect_temp() {
```

```
    echo "Signal"
    echo "Temperature"

    COUNT=$((TOTAL_DURATION / SAMPLING_INTERVAL))

    for ((i=0; i<=COUNT; i++)); do
        # Use `sensors` to fetch the temperature
        TEMP=$(sensors -u | grep -E 'temp[1-9]_input' | head -1 | awk '{print $2}')
        if [ -z "$TEMP" ]; then
            TEMP="NaN"
        else
            # Format the temperature to three decimal places
            TEMP=$(awk "BEGIN {printf \"%.3f\", $TEMP}")
        fi
        echo "$TEMP"
        sleep "$SAMPLING_INTERVAL"
    done
}



# Function to run a single experiment
run_experiment() {
```

```
    STRESS_NG_CMD="$1"
    PCM_FILENAME="$2"

    echo "Starting experiment with command: $STRESS_NG_CMD"
    echo "PCM output file: $PCM_FILENAME"

    # Start stress-ng in the background
    eval "$STRESS_NG_CMD" &

    # Start PCM data collection and redirect output to a temporary file
    PCM_TEMP_FILE="$(mktemp)"
    sudo "$PCM_DIR/pcm" /csv "$SAMPLING_INTERVAL" -i="$TOTAL_DURATION" 2>
"$OUTPUT_DIR/pcm_errors.log" > "$PCM_TEMP_FILE" &

    # Collect temperature data and redirect output to a temporary file
    TEMP_TEMP_FILE="$(mktemp)"
    collect_temp > "$TEMP_TEMP_FILE" &

    # Wait for both background processes to finish
    wait

    # Merge PCM data and temperature data
    PCM_LINES=$(wc -l < "$PCM_TEMP_FILE")
```

```
    TEMP_LINES=$(wc -l < "$TEMP_TEMP_FILE")

    # Ensure both files have the same number of lines
    if [ "$PCM_LINES" -ne "$TEMP_LINES" ]; then
        echo "Mismatch in number of lines between PCM and temperature data."
        echo "PCM lines: $PCM_LINES, TEMP lines: $TEMP_LINES"
        echo "Adjusting temperature data to match PCM data."

        # Calculate the difference in lines
        LINE_DIFF=$((PCM_LINES - TEMP_LINES))

        # Add empty lines to TEMP_TEMP_FILE if necessary
        if [ "$LINE_DIFF" -gt 0 ]; then
            for ((i=0; i<LINE_DIFF; i++)); do
                echo "" >> "$TEMP_TEMP_FILE"
            done
        fi
    fi

    # Merge the two files
    paste -d ',' "$PCM_TEMP_FILE" "$TEMP_TEMP_FILE" >
"$OUTPUT_DIR/$PCM_FILENAME"
```

```
    # Remove temporary files
    rm "$PCM_TEMP_FILE" "$TEMP_TEMP_FILE"

    # Wait for stress-ng to finish
    wait

    echo "Experiment completed: $STRESS_NG_CMD"
}

# Array of experiments
declare -a experiments=(
    # CPU Stress - All cores
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 0 --timeout
$TOTAL_DURATION|Linux0Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 10 --timeout
$TOTAL_DURATION|Linux10Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 20 --timeout
$TOTAL_DURATION|Linux20Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 30 --timeout
$TOTAL_DURATION|Linux30Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 40 --timeout
$TOTAL_DURATION|Linux40Static.csv"
```

```
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 50 --timeout
$TOTAL_DURATION|Linux50Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 60 --timeout
$TOTAL_DURATION|Linux60Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 70 --timeout
$TOTAL_DURATION|Linux70Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 80 --timeout
$TOTAL_DURATION|Linux80Static.csv"
    "stress-ng --cpu 0 --cpu-method matrixprod --cpu-load 90 --timeout
$TOTAL_DURATION|Linux90Static.csv"

    # CPU Stress - 2 threads
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 0 --timeout
$TOTAL_DURATION|Linux0Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 10 --timeout
$TOTAL_DURATION|Linux10Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 20 --timeout
$TOTAL_DURATION|Linux20Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 30 --timeout
$TOTAL_DURATION|Linux30Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 40 --timeout
$TOTAL_DURATION|Linux40Static2threads.csv"
```

```
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 50 --timeout
$TOTAL_DURATION|Linux50Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 60 --timeout
$TOTAL_DURATION|Linux60Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 70 --timeout
$TOTAL_DURATION|Linux70Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 80 --timeout
$TOTAL_DURATION|Linux80Static2threads.csv"
    "stress-ng --cpu 2 --cpu-method matrixprod --cpu-load 90 --timeout
$TOTAL_DURATION|Linux90Static2threads.csv"

    # CPU Stress - 4 threads
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 0 --timeout
$TOTAL_DURATION|Linux0Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 10 --timeout
$TOTAL_DURATION|Linux10Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 20 --timeout
$TOTAL_DURATION|Linux20Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 30 --timeout
$TOTAL_DURATION|Linux30Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 40 --timeout
$TOTAL_DURATION|Linux40Static4threads.csv"
```

```
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 50 --timeout
$TOTAL_DURATION|Linux50Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 60 --timeout
$TOTAL_DURATION|Linux60Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 70 --timeout
$TOTAL_DURATION|Linux70Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 80 --timeout
$TOTAL_DURATION|Linux80Static4threads.csv"
    "stress-ng --cpu 4 --cpu-method matrixprod --cpu-load 90 --timeout
$TOTAL_DURATION|Linux90Static4threads.csv"

    # CPU Stress - 6 threads
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 0 --timeout
$TOTAL_DURATION|Linux0Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 10 --timeout
$TOTAL_DURATION|Linux10Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 20 --timeout
$TOTAL_DURATION|Linux20Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 30 --timeout
$TOTAL_DURATION|Linux30Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 40 --timeout
$TOTAL_DURATION|Linux40Static6threads.csv"
```

```
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 50 --timeout
$TOTAL_DURATION|Linux50Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 60 --timeout
$TOTAL_DURATION|Linux60Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 70 --timeout
$TOTAL_DURATION|Linux70Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 80 --timeout
$TOTAL_DURATION|Linux80Static6threads.csv"
    "stress-ng --cpu 6 --cpu-method matrixprod --cpu-load 90 --timeout
$TOTAL_DURATION|Linux90Static6threads.csv"



    # Memory Stress
    "stress-ng --vm 2 --vm-bytes 1G --timeout $TOTAL_DURATION|LinuxMem1.csv"
    "stress-ng --vm 4 --vm-bytes 2G --timeout $TOTAL_DURATION|LinuxMem2.csv"

    # I/O Stress
    "stress-ng --hdd 2 --timeout $TOTAL_DURATION|LinuxIO1.csv"
    "stress-ng --hdd 4 --timeout $TOTAL_DURATION|LinuxIO2.csv"
)

# Main loop to run all experiments
for exp in "${experiments[@]}"; do
```

```
    # Parse the experiment string
    IFS='|' read -r STRESS_NG_CMD PCM_FILENAME <<< "$exp"

    # Run the experiment
    run_experiment "$STRESS_NG_CMD" "$PCM_FILENAME"

    # Optional: Add a short delay between experiments
    sleep 1
done

echo "All experiments completed."
```

```windowsAnalyze.py
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Set seaborn style
sns.set_style('whitegrid')

# Define base directory
base_dir = "Data/NewData/Windows"

# Define thread-specific files
thread_files = {
    '2 Threads': os.path.join(base_dir, 'Windows2threads.csv'),
    '4 Threads': os.path.join(base_dir, 'Windows4threads.csv'),
    '6 Threads': os.path.join(base_dir, 'Windows6threads.csv'),
    '8 Threads': os.path.join(base_dir, 'Windows8threads.csv')
}

# Function to load data
def load_data(file_path, num_threads_label):
    if not os.path.exists(file_path):
        print(f"File not found: {file_path}")
        return None

    print(f"Loading file: {file_path}")
```

```
    # Read the file, skipping the first row (datatype row) and identifying the correct
header
    df = pd.read_csv(file_path, skiprows=1)

    # Ensure the DateTime column exists
    if 'DateTime' not in df.columns:
        print(f"'DateTime' column not found in {file_path}. Ensure the correct header row is
specified.")
        return None

    # Convert DateTime column to datetime format
    df['DateTime'] = pd.to_datetime(df['DateTime'], errors='coerce')
    df.dropna(subset=['DateTime'], inplace=True)

    # Rename 'C0res%' to 'LoadPercent' for consistency
    if 'C0res%' in df.columns:
        df.rename(columns={'C0res%': 'LoadPercent'}, inplace=True)
        print("Renamed 'C0res%' to 'LoadPercent'")

    # Add thread label for identification
    df['NumThreads'] = int(num_threads_label.split()[0])

    return df
```

```
# Load and combine data
data_frames = []
for num_threads_label, file_path in thread_files.items():
    df = load_data(file_path, num_threads_label)
    if df is not None:
        data_frames.append(df)

# Combine all thread-specific data
if not data_frames:
    print("No data files loaded. Exiting.")
    exit()

all_data = pd.concat(data_frames, ignore_index=True)

# Check if the dataset is empty
if all_data.empty:
    print("Dataset is empty. Exiting.")
    exit()

# --- TEMP Analysis ---
```

```
# Filter data for TEMP analysis
TEMP_data = all_data[['LoadPercent', 'NumThreads', 'TEMP']].dropna()

# Average TEMP by thread count and load
avg_temp = TEMP_data.groupby(['NumThreads',
'LoadPercent'])['TEMP'].mean().reset_index()

# Plot average TEMP
plt.figure(figsize=(10, 6))

# Scatter plot for TEMP analysis
sns.scatterplot(
    data=avg_temp,
    x='LoadPercent',
    y='TEMP',
    hue='NumThreads',
    style='NumThreads',
    palette='coolwarm',
    s=100
)

# Add annotations for small LoadPercent values
for i, row in avg_temp.iterrows():
```

```
    if row['LoadPercent'] < 1:
        plt.text(row['LoadPercent'], row['TEMP'],
                 f"{row['LoadPercent']:.2f}",
                 fontsize=9, ha='center')

plt.title('Average TEMP by Threads and Load')
plt.xlabel('Load Percentage (%)')
plt.ylabel('TEMP (°C)')
plt.legend(title='Threads')
plt.grid(True)
plt.show()

# --- Additional Metrics: Correlation with Energy ---

# Investigate correlation between TEMP and energy consumption
correlation_data = all_data[['TEMP', 'Proc Energy (Joules)', 'LoadPercent',
'NumThreads']].dropna()

# Scatter plot: TEMP vs Energy
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=correlation_data,
    x='TEMP',
```

```
    y='Proc Energy (Joules)',
    hue='NumThreads',
    style='NumThreads',
    palette='coolwarm',
    s=100
)
plt.title('Processor Energy Consumption vs TEMP')
plt.xlabel('TEMP (°C)')
plt.ylabel('Energy Consumption (Joules)')
plt.legend(title='Threads')
plt.grid(True)
plt.show()

# --- Regression Analysis ---

# Prepare data for regression
regression_data = all_data[['LoadPercent', 'NumThreads', 'FREQ', 'TEMP', 'Proc Energy
(Joules)']].dropna()

# Features and target
X = regression_data[['LoadPercent', 'NumThreads', 'FREQ', 'TEMP']]
y = regression_data['Proc Energy (Joules)']
```

```
# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Polynomial regression
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X_scaled)

model = LinearRegression()
model.fit(X_poly, y)

# Predictions and metrics
y_pred = model.predict(X_poly)
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print(f"Polynomial Regression - RMSE: {np.sqrt(mse):.4f}")
print(f"Polynomial Regression - R-squared: {r2:.4f}")

# Plot Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y, y_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', label='Perfect Fit')
```

```
plt.xlabel('Actual Energy Consumption (Joules)')
plt.ylabel('Predicted Energy Consumption (Joules)')
plt.title('Actual vs Predicted Energy Consumption')
plt.legend()
plt.grid(True)
plt.show()

# Residuals plot
residuals = y - y_pred
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--', label='Zero Error')
plt.xlabel('Predicted Energy Consumption (Joules)')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.legend()
plt.grid(True)
plt.show()

print("Analysis complete.")
```