# Promoting Green Coding in VS Code with GCPM: A Power Model for Heterogeneous Processors

Amina Nasrin, Dayuan Chen, Noe Soto, Ziliang Zong
*Computer Science Dept.*, *Texas State University*, San Marcos, USA
aminanasrin@txstate.edu, dayuan@txstate.edu, noe_soto@txstate.edu, ziliang@txstate.edu

*Abstract*—**Software applications have significantly improved our lives, making computer programming increasingly common and necessary in many fields. However, the energy consumption of software is often overlooked. This gap exists due to the lack of accurate, convenient, and host-independent power consumption measurement technologies for software. We address this issue by developing GreenCode-PowerMeter (GCPM), a model-based power estimation technology for computer programs. GCPM features a decision tree followed by a third degree multi-variable polynomial regression model to provide precise power consumption measurements within software development environment. Based on real power consumption data from 39 popular processors, provided by the SPEC benchmark across various CPU utilization levels, GCPM depicts a mean absolute percentile error of only 0.075%, outperforms 14 existing power models, including the TDP based model and closely aligns to Running Average Power Limit (RAPL). By integrating GCPM into the VS Code IDE as GreenCode-PowerMeasurement extension, we enable measuring software power consumption directly within the development environment. With 14 million active users of VS Code and GCPM's high accuracy exceeding 90% compared to RAPL, requiring no sudo permission; this integration has enabled precise power analysis - simplifying the adaptation for developers towards energy optimized software development.**

*Index Terms*—**green computing, sustainability, machine learning, power estimation, energy measurement, energy saving, energy aware software**

## I. Introduction

Accurate, convenient, and host-independent power measurement is crucial for addressing energy surges, echoing the adage: "If you can't measure it, you can't optimize it.". Comprehensive power measurements help organizations track sustainability progress and assess the benefits of energy-efficient coding practices. In 2023, California passed laws for companies to disclose their greenhouse gas (GHG) emissions and climate-related financial risks [1]. However, developing a robust software power measurement technique has challenges. Firstly, the enriched development environment with numerous programming languages with plethora of syntax and semantics complicate the creation of a universal model. Secondly, unique power characteristics of processors and OS-specific commands to execute programs require significant consideration for a functional model. Thirdly, relying on matrices like Thermal Design Power (TDP) or Running Average Power Limit (RAPL) are unreliable. TDP is reportedly inaccurate, offering only peak power estimates rather than real power usage under varied conditions [2], [3] while leveraging RAPL for its high precision demands sudo permissions and customized code different hardware.

In this paper, we address this research gap by implementing GreenCode-PowerMeter (GCPM), an ML based real time power measurement methodology integrated within VS Code IDE. Specifically, we make the following contributions:

- Introduce VS Code extension of GCPM with enhanced accessibility to conveniently measure program's power consumption regardless of host architecture and OS
- Demonstrate GCPM's superior accuracy compared to RAPL and it outperforming 14 existing models including TDP based model
- Illustrate GCPM's versatility in supporting multiple programming languages
- Validate GCPM's effectiveness to facilitate the development of energy aware software development

By offering quantified energy consumption for various coding strategies, GCPM encourages developers shifting towards more sustainable programming practices.

An overview of proposed methodology is illustrated in Fig. 1. In 1(a), GCPM is deployed on a host computer. Upon deployment, it collects processor specifications by interacting with the OS, as shown in 1(b). These specifications include processor model, vendor, frequency, cores, memory size, OS version, and threads per core. The Decision Tree in 1(c) uses these parameters to identify the closest matching processor of "GCPM processor" (in 1(d)) in our dataset, as in Table II. GCPM processor's specifications and real-time CPU utilization, as shown in 1(e), are input into a trained multi-variable polynomial regression model in 1(f), providing real-time power consumption estimation in 1(g).

The rest of the paper is structured as follows: Section II reviews existing CPU power consumption measurement techniques and their challenges. Section III details the data collection for model training, validation, and testing, as well as our notion for selecting multi-variable polynomial regression. Section IV describes the deployment and performance evaluation of our proposed technique. Section V presents two case studies illustrating the practical application of proposed model. We conclude the paper with a summary and our perspectives on potential future developments in Section VI.

## II. Related Works

### A. TDP and RAPL

To disclose the power consumption of the processors, chip manufacturers offer metrics based on theoretical design specification. Additionally, they incorporate specific on-chip registers that record real-time power-related data.

Thermal Design Power (TDP) defines the maximum power consumption and was originally defined to design the heat dissipation capacity required by a component's cooling system, such as a CPU [4]. Simply summing all components' TDP can provide maximum power consumption. Specific

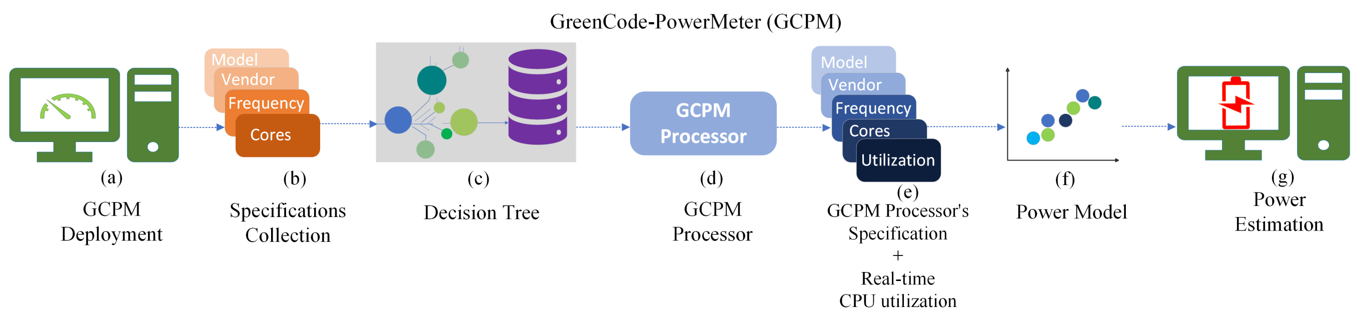


Fig. 1: Overall Workflow of GreenCode-PowerMeter (GCPM) - (a) GCPM being deployed on a Computer (b) GCPM collects host specifications for the Decision Tree (c) Decision Tree matches host specifications with dataset and outputs 'GCMP instance' (d) GCPM instance is the instance from the dataset that matches the closest to the host (e) GCPM Instance's specifications and real-time CPU utilization is collected for Power Model (f) Multi-variable Polynomial Regression Power Model estimates power consumption from (e), (g) Power Estimation is achieved

utilization power can be calculated by multiplying the TDP with the corresponding utilization. Despite its simplicity, TDP has been reported to overestimates up to 50% more power consumption [2]. This discrepancy arises because TDP represents the highest achievable power consumption under maximum theoretical load, which is unattainable in reality, even when the CPU is 100% utilized.

Running Average Power Limit (RAPL) is a feature in modern Intel and AMD processors that allows for monitoring the power consumption of various components within the CPU and is repudiated for it's high precision. RAPL requires root permissions to utilizes Model Specific Registers (MSR). Hence, it's implementation is restricted on environments like cloud, distributed servers and centralized database because improper use of MSR can lead to system instability or security compromise.

## B. Software Based Power Models

The alternative of physical measurement is software-based power modeling, which estimates the power consumption of the CPU based on software-level metrics and hardware specifications. These models typically utilize various performance affecting parameters, such as CPU frequency, instructions per cycle, cache misses, and other architectural features, to predict power consumption.

Fan et al. [5] introduced model based power consumption estimation procedure (we labeled as PM1) that uses power consumption at maximum CPU utilization, $P_{max}$ and power consumption when the CPU is idle, $P_{min}$ to estimate the power consumption, P at a certain CPU utilization, $u_{cpu}$. Anton et al. [6] asserted that $P_{min}$ of a server is nearly 70% of the value of $P_{max}$ . Subsequently, Han et al. came up with supporting studies of PM2 [7]. The interpolation based linear model has proved to be effective on power estimation on cloud simulation platforms [2], [8]. Ramya et al. introduced the concept of utilization based Linear Regression derived from [9] and provided PM4 [10]. Here $\alpha$ is the intercept of the regression and $\beta$ is the regression coefficient. These values are calibrated while training the model. Later, Zhikui et al. modeled the dynamic thermal model for CPU temperature and improvised the linear function of PM4 into PM5 [12]. Janacek et al. [13] introduced the power consumption of a server changes accordingly with the squared of the CPU frequency. This latest discovery was plausible due to

TABLE I: Existing Power Models

| Name | Power Model | References |
|---|---|---|
| PM1 | $P = P_{min} + (P_{max} - P_{min}) \cdot u_{cpu}$ | [5], [11] |
| PM2 | $P = 0.7 \cdot P_{max} + 0.3 \cdot P_{max} \cdot u_{cpu}$ | [6], [7] |
| PM3 | $P = P_1 + \frac{(u_{cpu} - u_{cpu_1}) \cdot (P_2 - P_1)}{u_{cpu_2} - u_{cpu_1}}$ | [2], [8] |
| PM4 | $P = \alpha + \beta \cdot u_{cpu}$ | [9], [10] |
| PM5 | $P = P_{min} + \beta \cdot u_{cpu}$ | [12] |
| PM6 | $P = \alpha + \beta_1 \cdot u_{cpu} + \beta_2 \cdot u_{cpu}^2$ | [13] |
| PM7 | $P = \alpha + \beta_1 \cdot u_{cpu} + \beta_2 \cdot u_{cpu}^2 + \beta_3 \cdot u_{cpu}^3$ | [14]–[16] |
| PM8 | $P = \alpha + \beta_1 \cdot u_{cpu} + \beta_2 \cdot u_{cpu}^r$ | [14], [15] |
| PM9 | $P = f(x) = W \cdot \phi(x) + b$ | [17] |
| PM10 | $P = \phi(W \cdot x + b)$ | [18] |
| PM11 | $P = \phi(\cdot) = \{x_i^a : 1 \leq a \leq 3\}$ | [14], [19] |
| PM12 | $P = \underset{\beta}{\text{argmin}} \sum_{n=1}^{N} \frac{1}{2}(y_n - \beta x_n)^2 + \lambda \sum_{i=1}^{p} \beta_i^2$ | [19], [20] |
| PM13 | $P(x) = \frac{1}{1+e^{-(\frac{x-\mu}{s})}}$ | [21], [22] |

the recent development of Dynamic Voltage and Frequency Scaling (DVFS) on processor architecture. So, he increased the degree of CPU utilization to 2 and included further parameters to the equation in PM6. Micha et al. [16] proposed PM7. In PM8, $r$ is the model parameter calibrated through regression. PM8 was introduced by Guazzone et al. [23] to provide power models with quadratic degree. PM9 is based on Support Vector Machine (SVM). Instead of fitting into a curve like the previous models, SVM aims at fitting the training data points into a hyperplane. Hence, SVM performs efficiently where large number of dimensions is involved [17]. In order to leverage the Multi Layer Perceptron(MLP), Leandro et al. proposed Artificial Neural Network (ANN) based Power Model in order to address the non-linear relation between power consumption to CPU utilization [18]. In ANN model, each layer derives output by computing the equation provided in the table for PM10. ANN assigns the weight of each layer as $W$, input vector as $i$, and bias vector as $b$. PM11 is the polynomial function with Lasso Regression which was introduced for reducing the number of regression variables. Lasso uses L1 regularization to obtain a subset of regression variables [14], [19]. In Ridge regression (PM12), $\beta$ is constrained to be limited within the hyper sphere around the center, by optimizing the Residual Sum of Squares (RSS). Here $\beta$ is expressed as function representing the maximum a posteriori (MAP) estimation [19], [20]. Logistic Regression (PM13), the sigmoid function is used to estimate the achieving a certain vaule of power during utilization [21], [22]. Here, $s$ is the scale parameter, $\mu$ is the mean value and $x$ is the power affecting input parameters like utilization and specifications.

However, the performance accuracy of ML based power models largely depends on the accuracy of the data these models are trained on. In Section III-A, we focus on collecting precise processor power consumption data across various workloads, and in Section III-B, we explore and apply the above mentioned techniques to propose an effective power consumption measurement method.

## III. Data Acquisition and Model Selection

In this section, we outline how we collect power data and select the power model in our study. First, we identify a range of computer processors that are used in personal computers, collect power consumption data from these processors to train our models. Following this, we perform a comparative analysis of various existing power models to evaluate their performance. This analysis enables us to select the most suitable model for accurate power estimation.

### A. *Collection of Power Data through SPEC Benchmark*

Collection of reliable processor power consumption data is crucial for the power model's performance. In order to collect reliable processor power consumption data, we leverage the SPEC benchmark [24], which provides comprehensive and accurate performance measurement of wide range of processors. SPEC captures power data for 11 different workloads, ranging from active idle (less than 10%) to 100% (maximum CPU utilization), typical of processor power consumption studies.

From the extensive list of processors available on SPEC benchmark, we enlist commonly used processors over past years. We locate 39 unique processors shown in Table II, including the processor manufacturer, processor model, core count, and operating frequency. After combining these data with their power consumption data and corresponding workloads, our dataset contains 429 rows representing the power consumption of these 39 different processor specifications. Our dataset features a range of processors, including Intel Xeon E series, L Series, Pentium D, Core i3, as well as AMD Opteron 8000 series, 2000 Series, 4000 Series processors and ARM Cortex-A53. These processors vary in number of cores ranging from 2 - 24, in frequency from 1.0GHz to 3.7GHz, and with some of these hyperthread enabled.

### B. *Power Model Selection*

To initiate our experimentation, we divide our dataset sequentially into training and test sets at a 75:25 ratio, with 75% for training and 25% kept unseen for testing. We further partition our training dataset randomly into a training set and a validation set using a 65:35 ratio to ensure optimal regression output.

TABLE II: Cloud VM instances and their specifications

| Processor Manufacturer | Processor Model | Core Count | Threads | Frequency (GHz) |
|---|---|---|---|---|
| Intel | E-2356G | 6 | 12 | 3.2 |
| | E-2388G | 8 | 16 | 3.7 |
| | E-2176G | 6 | 12 | 3.7 |
| | E3-1230 | 4 | 8 | 3.5 |
| | E3-1260L | 4 | 8 | 2.9 |
| | E3-1240L | 4 | 8 | 2.1 |
| | E3-1270 | 4 | 8 | 3.4 |
| | E3-1275L | 4 | 8 | 2.7 |
| | E5-2620 | 12 | 24 | 2.0 |
| | E5-2609 | 8 | 8 | 2.4 |
| | E3-1280 | 4 | 8 | 3.6 |
| | E3-1265L | 4 | 8 | 2.5 |
| | E5-2470 | 8 | 16 | 2.3 |
| | E5-2640 | 12 | 24 | 2.5 |
| | X-3470 | 4 | 8 | 2.93 |
| | X-3480 | 4 | 8 | 3.07 |
| | Core i3-540 | 2 | 4 | 3.07 |
| | E-3110 | 2 | 2 | 3.0 |
| | Pentium D-930 | 2 | 2 | 3.0 |
| | Xeon-3040 | 2 | 2 | 1.86 |
| | 3475 | 2 | 2 | 2.66 |
| | 7020 | 8 | 16 | 2.66 |
| | 7110M | 8 | 16 | 2.6 |
| | 3040 | 2 | 2 | 1.86 |
| | 3075 | 2 | 2 | 2.66 |
| | 3080 | 4 | 8 | 3.07 |
| | L3360 | 4 | 4 | 2.83 |
| | L5408 | 8 | 8 | 2.13 |
| | L5430 | 8 | 8 | 2.67 |
| AMD | 4344P | 8 | 16 | 3.8 |
| | 8024PN | 8 | 16 | 2.05 |
| | 8124PN | 16 | 32 | 2.0 |
| | 8124P | 16 | 32 | 2.45 |
| | 8024P | 8 | 16 | 2.4 |
| | 4310 | 8 | 8 | 2.2 |
| | 4164EE | 6 | 6 | 1.8 |
| | 2376HE | 8 | 8 | 2.3 |
| | 2384 | 8 | 8 | 2.7 |
| ARM | Cortex-A53 | 24 | 24 | 1.0 |

Performance analysis of the power models based on mean absolute percentage error is presented in Fig. 2 on training, validating, and testing. PM1-PM3 do not have any training and validation error as these are not regression based models rather have fixed slopes and intercepts. PM4, demonstrates consistent high errors across all phases; followed by PM9, PM12, PM13. PM2 shows the second highest error. PM11 does not perform well on unseen dataset as does not PM10, suggesting limitations in their predictive accuracy. The performance of PM5 - PM8 is consistent. However; PM1, PM7, and PM8 demonstrates error below 0.1 throughout training, validation, and testing phases. Given their superior performance, we conduct a detailed analysis of these models to determine the most effective one. To delve deeper into performance trends, we analyze the error for PM1, PM7, PM8 across different workloads during test, in Fig. 3.

PM1 depicts inconsistent performance by showing low error for very low (>20%) and very high (<80%) CPU utilization and significantly higher error rates from 20%-80% CPU utilization compared to the other three models. In contrast, both PM7 and PM8 has consistent performance. Although PM8 has error rate slightly lower than PM7, PM7 outperforms PM8 in runtime by half with 0.0051 seconds for

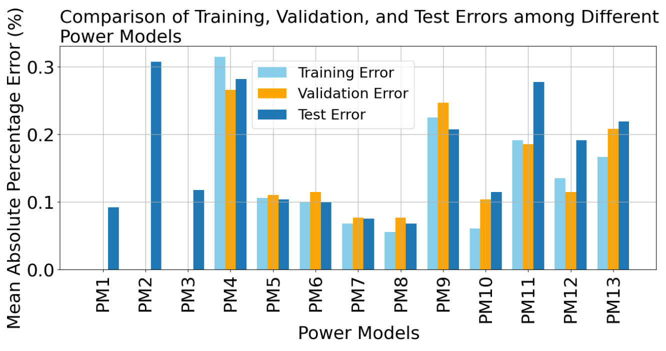


Fig. 2: Mean Absolute Percentage Error of Power Models

PM7 and 0.020 seconds for PM8. Hence, we choose PM7 to develop the model with minimal overhead.

### C. *Decision Tree*

The rapidly evolving world of computer processor architectures requires a robust methodology to ensure accuracy in power consumption estimation by tackling the processor-specific nature of power models To address these challenges, we introduce our methodology to Decision Tree. We systematically select power consumption affecting parameters of a processor for the Decision Tree:

- number of cores
- processor frequency
- processor manufacturer

Upon deployment on a previously unseen processor, Decision Tree navigates through the above mentioned parameters of the host processor and tries to locate the closest match processor from our dataset in Table II. We call this closest match processor “GCPM Processor”. Fig. 4 presents an example processor of Intel E5-2460 at 2.3GHz frequency with 8 cores. This is unknown for GCPM so far. The Decision Tree proceeds by selecting following decison nodes - 8 as Number of Cores, 2.3 as frequency, Intel as processor vendor thus reaching E5-2470 of Decision Node as GCPM Processor.

### D. *GreenCode-PowerMeter VS Code Extension*

One of the major obstacles of lowering software power consumption is the inconvenience in power measurement across development environment. Hence, we integrate GCPM

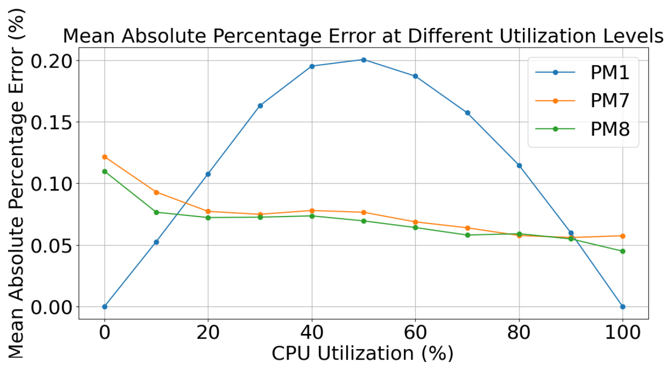
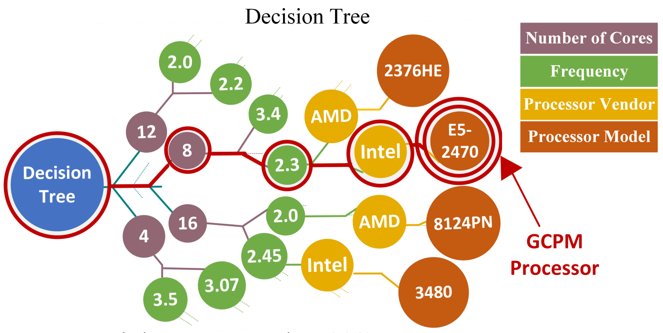


Fig. 3: Mean Absolute Percentage Error of the three top performing PM1, PM7, PM8 during testing



Example host specs - Intel E5-2460, 2.3GHz frequency, 8 cores

Fig. 4: Decision Tree Locating GCPM Processor - for example host specs, Decision Node of 8 selected as Number of cores, 2.3 as Frequency, Processor Vendor selected as Intel, Processor Model E5-2470 - finally reaching leaf node as GCPM Processor

with the most popular IDE of VS Code (with 14millions users) [25] by converting it into a VS Code extension.

Once installed, GCPM can be launched from context menu or command pallet. Upon launch, GCPM starts logging real-time power consumption by following the steps in Fig. 1 at a frequency of 10 power estimation per second while waiting for the input from user. A sample of input commands for different programming languages is presented in Table III. GCPM takes the command to run a program as input in a prompt provided upon launch. As soon as the program execution finishes, GCPM presents the power consumption for the duration of the program run in a csv files, in the same location as the run program. GCPM is hardware and OS independent and is observed to perform estimation for programs written in C, C++, Java, Python. Detailed experiments supporting this statement is provided in Section IV and V.

## IV. Experimental Result

In this section, we evaluate GCPM’s performance against RAPL model. Our host consists of Intel Core i5-6500 processor with 4 core, 4 threads and 3.60GHz frequency, running Ubuntu 22.04. This processor is unknown for GCPM hence decision tree is crucial here. We have the privilege of running RAPL on this machine. We start observing performance at idle state and then begin to deploy load that starts by running the processor at 10% CPU utilization and gradually increasing by 10%, all the way to 100% CPU utilization. We set the power consumption estimation interval at 100ms. From observations, it is revealed that our model effectively captures fluctuations in CPU utilization and estimates power

TABLE III: Different Input Commands for GCPM Input for helloworld program

| Programming Language | Command to pass in GCPM input box |
|---|---|
| Python | python/python3 filepath/helloworld.py |
| C | filepath/./helloworld |
| C++ | filepath/./helloworld |
| Java | java filepath/hellowrold |
| JavaScript | filepath/helloworld.js |

consumption, as illustrated in Fig. 5. As we observe, GCPM successfully adapts seamlessly to the new environment, handling varying workloads without requiring administrative privileges and incurring minimal overhead. As we observe, GCPM successfully performs on the Intel processor and captures real-time power consumption closely lying to RAPL (nearly overlapping for 100% CPU utilization) which implies the high accuracy of GCPM.

Fig. 5 also compares the performance of the GCPM with the traditional TDP based model. We see that GCPM's power estimation varies largely compared to TDP and TDP based model while keeping close approximation to RAPL. Numerically, with respect to RAPL - GCPM's mean absolute error is 1.63 whereas TDP based model's 19.331. Hence, it is safe to state that GCPM outperforms TDP and TDP based model. GCPM's performance enhencement compared to TDP or TDP-based model can be credited to it mapping the host processor to the processors in Table II using a decision tree, and the third-degree multi-variable polynomial model effectively capturing all power-affecting parameters.

## V. Case Studies

We demonstrate the usefulness of GCPM using the computer programs of Shell Sort and Fast Fourier Transformation (FFT). We quantify their power consumption using GCPM and implicate that using different programming approaches significantly reduce program's energy consumption without compromising performance. This way, GCPM is advantageous to develop energy optimized software.

### A. *Quantifying Energy Consumption of Shell Sort algorithm using GCPM*

We develop a basic version program of Shell Sort algorithm in python programming language that sorts input size of 1 million data. We use GCPM while running this program to quantify the energy consumed by this program. The experiment environment consists of AMD 3250U processor with 2 physical cores and 2 threads per core at 2.60GHz frequency running Windows 11. This processor is new to GCPM hence leverages the Decision Tree. The program runs for 0.92 seconds and GCPM successfully generates the power report in a csv file for the basic version of the Shell Sort program. From the reported power estimations, we calculate the energy consumption of 26.49J.

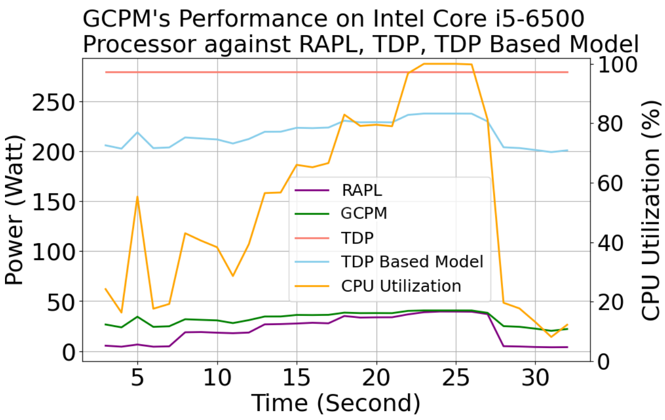


Fig. 5: Performance Comparison of GCPM on Intel Core i5-6500 in Real-Time Power Consumption Estimation, RAPL, TDP, and TDP Based Model; GCPM Estimates Real-Time Power Consumption Closest to the RAPL and outperforms both TDP and TDP Based Model

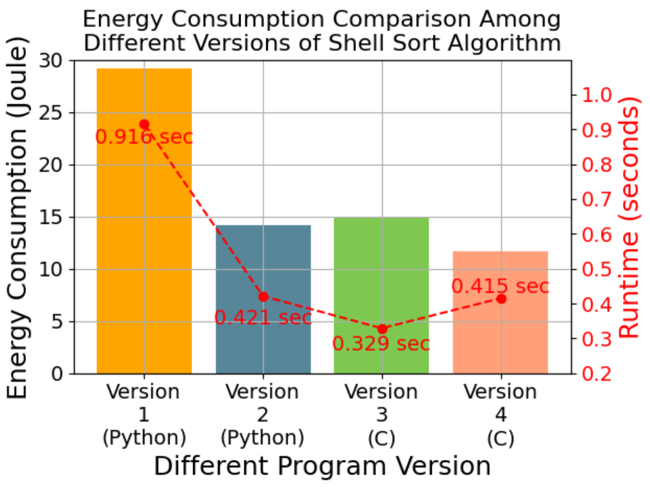


Fig. 6: Energy consumption of Shell Sort program is optimized in different versions after quantification using GCPM. The bar plots represents the energy consumption in Joule and the red dotted line ( - - - ) represents the runtime in seconds.

Next, we develop 3 versions of the program with energy optimization - one in Python, and two in C. Optimization approaches include using optimized gap sequence, combining comparison and assignment operation in the same loop, asynchronously outputting sorted array. Gaped insertion reduces cache misses by initially moving elements at larger distance, subsequently positioning them in closer to appropriate positions. In case of C; compiling the program before running, significantly reduces energy consumption. We plot the energy consumption in of all these four versions of the program into Fig. 6 and observe a 60.12% energy saving attained in the optimized C version compared to the first developed Python version.

However, the seemingly proportionate relationship between runtime and energy consumption might boast the misconception of "A faster program is an energy efficient program". Our next demonstration implicates the opposite - to attain energy efficiency program does not need to run faster.

### B. *Energy Efficiency vs. Execution Speed in Software Optimization*

A misleading intuition in software energy optimization is "A faster program is an energy efficient program". In this subsection, we execute an FFT program at two different runtime (lower runtime at maximum CPU frequency and higher runtime at minimum CPU frequency) and quantify their power and energy consumption using GCPM for different input sizes to verify this assumption. We take input sizes of 5 hundred thousand, 1 million, and 2 million. As in Fig. 7, a reverse proportionate relation is observed between energy consumption and runtime. For each case, the energy consumption is lower at longer runtime. This implies that, there are scenarios when faster execution of a program does

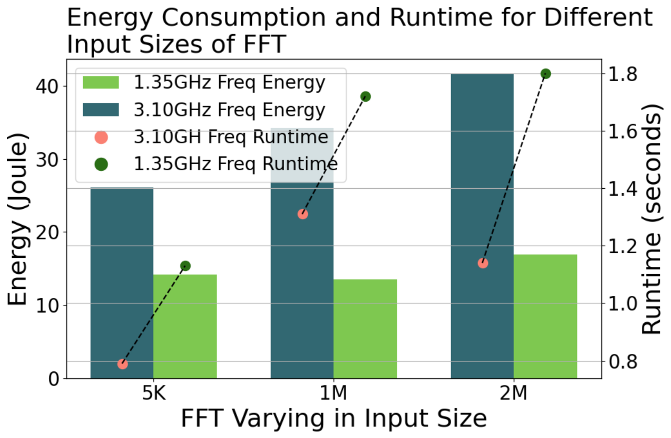


Fig. 7: Energy Consumption quantified using GCPM for FFT with different input sizes running at maximum and minimum CPU frequency to incur respective lower and higher runtime. Energy consumption at minimum runtime in each case is higher than energy consumption at maximum runtime

not guarantee the energy efficiency. In order for ensuring energy efficiency, programmers indeed require to directly quantify program's energy consumption using technology like GCPM and take actions accordingly.

## VI. Conclusion and Future Plan

Accurate power measurement in software development is crucial for contributing to comprehensive energy goals. We propose GCPM, integrating decision tree and third-degree polynomial regression, trained with real power consumption data from SPEC benchmarks. The decision tree ensures versatility on unfamiliar processors. Integration into VS Code enhances convenience. GCPM's 90% performance accuracy compared to RAPL ensures its reliability. While evaluating GCPM's effectiveness, we observe a 28% energy saving with visual quantification by examining different programming approaches, highlighting greater energy-saving opportunities.

However, GCPM currently only estimates processor power consumption. In the future, we plan to include GPU and memory power consumption in GCPM to provide an overall power estimation for specific program runs.

## Acknowledgement

We thank SPEC for benchmarking support and providing diverse processor power consumption data. This collaboration has advanced our methodologies in computational sustainability and power measurement. We extend our sincere appreciation everyone who assisted in testing the extension and ensuring its reliability.

## References

[1] C. Legislature, "The climate corporate data accountability act," 2023, s.B. 253, 261.

[2] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach.* Elsevier, 2011.

[3] R. Schöne, T. Ilsche, M. Bielert, M. Velten, M. Schmidl, and D. Hackenberg, "Energy efficiency aspects of the amd zen 2 architecture," in *2021 IEEE International Conference on Cluster Computing (CLUSTER).* IEEE, 2021, pp. 562–571.

[4] D. Ganapathy and E. J. Warner, "Defining thermal design power based on real-world usage models," in *2008 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems.* IEEE, 2008, pp. 1242–1246.

[5] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH computer architecture news*, vol. 35, no. 2, pp. 13–23, 2007.

[6] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems*, vol. 28, no. 5, pp. 755–768, 2012.

[7] G. Han, W. Que, G. Jia, and L. Shu, "An efficient virtual machine consolidation scheme for multimedia cloud computing," *Sensors*, vol. 16, no. 2, p. 246, 2016.

[8] R. Kress, *Numerical analysis.* Springer Science & Business Media, 2012, vol. 181.

[9] J. A. Baglivo, *Mathematica laboratories for mathematical statistics: Emphasizing simulation and computer intensive methods.* SIAM, 2005.

[10] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No" power" struggles: coordinated multi-level power management for the data center," in *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, 2008, pp. 48–59.

[11] T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini, "Mercury and freon: temperature emulation and management for server systems," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5, pp. 106–116, 2006.

[12] Z. Wang, N. Tolia, and C. Bash, "Opportunities and challenges to unify workload, power, and cooling management in data centers," in *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, 2010, pp. 1–6.

[13] S. Janacek, K. Schröder, G. Schomaker, W. Nebel, M. Rüschen, and G. Pistoor, "Modeling and approaching a cost transparent, specific data center power consumption," in *2012 International Conference on Energy Aware Computing.* IEEE, 2012, pp. 1–6.

[14] L. Ismail and H. Materwala, "Computing server power modeling in a data center: Survey, taxonomy, and performance evaluation," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–34, 2020.

[15] A. Tarafdar, S. Sarkar, R. K. Das, and S. Khatua, "Power modeling for energy-efficient resource management in a cloud data center," *Journal of Grid Computing*, vol. 21, no. 1, p. 10, 2023.

[16] M. vor dem Berge, G. Da Costa, A. Kopecki, A. Oleksiak, J.-M. Pierson, T. Piontek, E. Volk, and S. Wesner, "Modeling and simulation of data center energy-efficiency in coolemall," in *Energy Efficient Data Centers: First International Workshop, E 2 DC 2012, Madrid, Spain, Mai 8, 2012, Revised Selected Papers 1.* Springer, 2012, pp. 25–36.

[17] L. Luo, W. Wu, W.-T. Tsai, D. Di, and F. Zhang, "Simulation of power consumption of cloud data centers," *Simulation Modelling Practice and Theory*, vol. 39, pp. 152–171, 2013.

[18] L. F. Cupertino, G. Da Costa, and J.-M. Pierson, "Towards a generic power estimator," *Computer Science-Research and Development*, vol. 30, pp. 145–153, 2015.

[19] D. M. Blei, "Regularized regression," 2015, columbia University, December 15, 2015. [Online]. Available: https://www.cs.columbia.edu/ blei/fogm/2015F/notes/regularized-regression.pdf

[20] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2009, vol. 2.

[21] M. Daraghmeh, S. B. Melhem, A. Agarwal, N. Goel, and M. Zaman, "Linear and logistic regression based monitoring for resource management in cloud networks," in *2018 IEEE 6th international conference on future internet of things and cloud (FiCloud).* IEEE, 2018, pp. 259–266.

[22] M. B. Issa, M. Daraghmeh, Y. Jararweh, M. Al-Ayyoub, M. Alsmirat, and E. Benkhelifa, "Using logistic regression to improve virtual machines management in cloud computing systems," in *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS).* IEEE, 2017, pp. 628–635.

[23] M. Guazzone, C. Anglano, and M. Canonico, "Exploiting vm migration for the automated power and performance management of green cloud computing systems," in *Energy Efficient Data Centers: First International Workshop, E 2 DC 2012, Madrid, Spain, Mai 8, 2012, Revised Selected Papers 1.* Springer, 2012, pp. 81–92.

[24] Standard Performance Evaluation Corporation, "Spec power and performance benchmark," https://www.spec.org/power_ssj2008/results/, accessed: 2024-07-03.

[25] Microsoft, "Visual studio code," https://code.visualstudio.com/, accessed: 2024-07-09.