DOI: 10.15514/ISPRAS-2025-37(4)-16

Using Software-Defined Performance Counters to Construct a GPU Power Consumption Model

A.N. Gulin, ORCID: 0009-0001-0519-0101 <gulin-2001@mail.ru>
S.M. Staroletov, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>

Polzunov Altai State Technical University
46, prospect Lenina, Barnaul, Altai region, 656038, Russia.

Abstract. In the modern world, processor performance and energy efficiency play a key role in computer system design. Along with CPUs, GPUs are powerful computing devices used for computer graphics processing, machine learning, and more. Processors are equipped with built-in sensors accessible through specialized tools. The chip of a modern video card can operate in a fairly wide range of frequencies and power limits (PLs). Very often, when solving a computational task or rendering a scene, the video card can operate more optimally, without wasting excess power, which can significantly save energy on labor-intensive tasks. Therefore, it is important for a set of given tasks to find such parameters where the ratio of useful work per watt will be maximum. After conducting a large number of experiments, one can learn to predict the dependence of such a target function on the parameters. This paper examines obtaining current GPU parameter values using various tools. We present results of collecting raw data from NVIDIA GPUs and the subsequent construction of an optimal power consumption model.

Keywords: GPU; NVAPI; power consumption model; GPU sensors.

For citation: Gulin A.N., Staroletov S.M. Using software-defined performance counters to construct a GPU power consumption model. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025., pp. 17-30. DOI: 10.15514/ISPRAS-2025-37(4)-16.

Использование программно-определяемых счетчиков для построения модели оптимального энергопотребления графического процессора

А.Н. Гулин, ORCID: 0009-0001-0519-0101 <gulin-2001@mail.ru>
С.М. Старолетов, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>

АлтГТУ им. И.И. Ползунова
Россия, 656038, Алтайский край, г. Барнаул, пр. Ленина, 46.

Аннотация. В современном мире производительность и энергоэффективность процессоров играют ключевую роль при разработке компьютерных систем. Наряду с CPU, GPU являются мощными вычислительными устройствами и используются для обработки компьютерной графики, машинного обучения и многого другого. Процессоры оборудованы встроенными датчиками, доступ к которым осуществляется через специальные средства. Чип современной видеокарты может работать в достаточно широком диапазоне частот и лимитов мощности (PL). Очень часто при решении вычислительной задачи или рендеринге сцены видеокарта может работать более оптимально, без затрат избыточной мощности, что позволяет значительно экономить энергию при выполнении ресурсоёмких задач. Поэтому для заданного набора задач важно найти такие параметры, при которых отношение полезной работы к затраченной мощности будет максимальным. После проведения большого количества экспериментов можно научиться прогнозировать зависимость данной целевой функции от параметров. В данной статье рассматривается получение текущих значений параметров GPU при помощи различных инструментов. Представлены результаты сбора сырых данных с видеокарт NVIDIA и последующего построения модели оптимального энергопотребления.

Ключевые слова: графический процессор; NVAPI; модель энергопотребления; датчики видеокарты.

Для цитирования: Гулин А.Н., Старолетов С.М. Использование программно-определяемых счетчиков для построения модели оптимального энергопотребления графического процессора. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 17-30 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4)– 16.

1. Introduction

GPU (graphics processing unit) is designed to accelerate graphics rendering and parallel computations. The term “GPU” was first introduced by NVIDIA in 1999 with the release of GeForce 256. This was the first graphics card marketed as a GPU since it integrated transformation, lighting, and rendering mechanisms on a single chip, while previous graphics cards used the central processor for some of these tasks. Even in the early days of graphics cards, manufacturers transitioned to digital power management [1].

GPU differs from CPU in its architecture optimized for parallel computing. Unlike CPUs with several powerful cores, GPUs consist of thousands of simpler computing units grouped into clusters. These units are organized into streaming multiprocessors, each capable of executing multiple threads simultaneously. This structure allows GPUs to efficiently process large data volumes, such as pixels in graphics or matrices in machine learning [2].

GPUs achieve high performance computing through massive parallelism called SIMT (Single Instruction, Multiple Threads). For example, in rendering, one shader (a GPU worker) can process thousands of pixels in parallel. Additionally, GPUs use warps (NVIDIA) or wavefronts (AMD) – thread groups executing one instruction synchronously, minimizing thread management overhead [2-3].

A key aspect of GPU architecture is its complex memory hierarchy. Global memory (VRAM) has high bandwidth but significant latency, so GPUs employ multiple cache levels and registers to accelerate data access. Each streaming multiprocessor (SM) has its own shared memory enabling low-latency data exchange between threads within one block. Such process can be modeled [4-5].

Modern GPUs also use unified memory technology allowing CPUs and GPUs to work with shared address space, simplifying programming [2].

By automatically adjusting voltage and frequency parameters for the graphics processor and video memory according to GPU’s BIOS algorithms, the graphics card attempts to maintain maximum frequency and performance without exceeding maximum permissible power consumption, typically specified by TDP and TGP parameters. The TDP (Thermal Design Power) parameter for graphics cards indicates the maximum heat the cooling system must dissipate [6]. The TGP (Total Graphics Power) parameter specifies total power consumption of the graphics processor, PCB, and video memory, closest to actual card consumption. Due to increased power supply requirements, power consumption must be strictly limited. The Power Limit (PL) parameter prevents exceeding calculated consumption. Adjusting PL downward can balance low power consumption with acceptable temperatures and performance [7]. When modern graphics cards reach power limits, mechanisms reduce core frequency and voltage to maintain specified consumption. Proper undervolting through voltage reduction minimally impacts performance. Excessive voltage reduction may cause system instability and crashes, requiring constant GPU temperature and performance monitoring [8]. Undervolting is completely safe: at critical voltage levels the system simply reboots.

Present work aims to study methods for obtaining raw GPU sensors data for subsequent analysis, examine existing tools for collecting such data and power consumption estimation, develop an optimized GPU power consumption model, and present results from experiments on actual hardware.

The paper is structured as follows. Section 2 reviews existing work on GPU and CPU data collection and analysis. Section 3 describes various tools for collecting GPU sensor data. Section 4 presents the developed solution for GPU data collection, model construction, as well as experimental results. The Conclusion summarizes the findings.

The present paper is an extension of the report presented at SYRCoSE Software Engineering Colloquium 2025 in Pyatigorsk.

2. Related works

This section reviews works related to obtaining and estimating power consumption for both GPUs and CPUs.

PowerAPI, described in [9], is a software package providing CPU power consumption information based on sensor data. The system comprises multiple components running as Docker containers or standalone applications (PowerAPI components can run via Python). For CPU data collection, it uses HWPC – a hardware performance counter utilizing RAPL technology for CPU monitoring. The HWPC sensor records raw data in MongoDB. The SmartWatts "formula" application then processes this data to produce power estimates. InfluxDB, specialized for time-series data, stores values computed by SmartWatts [10]. As of 2024, PowerAPI (particularly HWPC) only supports Linux and limited CPU architectures (Intel Sandy Bridge or newer, excluding Intel Core Tiger Lake, Alder Lake, Raptor Lake; AMD Zen 1, 2, 3, 4).

PowerAPI's modeling approach has two phases: training and deployment. The training phase analyzes power consumption and runs target processor events to identify those most impacting consumption. These key events form an energy model – a formula used during online deployment for real-time power estimation. The data allows PowerAPI to build a power consumption model enabling efficient real-time energy cost determination. To reduce error between predicted and actual results, models are computed for workloads with lowest average Pearson coefficient across all HWPC event combinations. Then, for each workload, the model with smallest error is selected, and from these, one model minimizing error across all workloads is chosen [11].

An Intel Xeon W3520 (Bloomfield family) running Ubuntu 14.04 (kernel 3.13) was tested with standard configuration. Power consumption estimates were obtained for parallel processes running on the same CPU, specifically power distribution between idle consumption, the "freqmine" benchmark from PARSEC, and two other NPB benchmarks (bt.C and cg.C). Compared to physical PowerSpy measurements at 4 Hz (one power estimate per 250 ms), PowerAPI achieved 2% relative error (2.92 W) [12].

PowerAPI limitations include no GPU power modeling support, Linux-only operation, and limited CPU architecture support. Another drawback is system complexity comprising multiple separate components (requiring Docker installation).

PyJoules, described in [13], is an open-source Python toolkit (library) for measuring host machine energy consumption during Python code execution, available on GitHub and derived from "PowerAPI". PyJoules tracks energy consumed by specific host devices: socket packages for Intel CPUs, RAM (for Intel server architectures), Intel integrated graphics (client architectures), and NVIDIA GPUs. PyJoules uses Intel's "Running Average Power Limit" (RAPL) technology estimating CPU, RAM, and integrated GPU consumption, available on Intel processors since Sandy Bridge (2010).

Currently PyJoules uses Linux kernel APIs to access RAPL-reported energy values, making CPU/RAM/iGPU monitoring unavailable on Windows or MacOS. As RAPL isn't exposed in VMs, PyJoules can't monitor consumption within virtual machines.

PyJoules limitations include restricted system support (Linux only, no VMs), no AMD GPU support, no optimal power modeling capability, and inability to track overall system consumption rather than just monitored code segments. Formally, measurements are affected by all running processes, but tracking their consumption requires monitoring stub functions.

The SAOU framework in [14] combines overclocking (exceeding safe maximum frequency) and undervolting (reducing voltage below safe levels) for GPUs, achieving up to 22% energy savings without performance loss. The method uses enhanced checkpoint-recovery to correct errors from extreme settings. Testing with cuBLAS matrix multiplication showed SAOU dynamically selects optimal voltage/frequency levels based on error models. On NVIDIA GTX 980, gradual frequency increases yielded 5.3% power reduction versus baseline.

SAOU limitations include limited testing (only cuBLAS-MM), unverified effectiveness for other workloads, requiring Linux kernel modifications and GPU thread synchronization, support for GPU dynamic frequency/voltage management, additional resources for recovery systems, and prerequisite profiling for safe frequency/voltage determination.

The GreenMM framework in [15] enables energy-efficient matrix multiplication on GPUs via undervolting using algorithmic fault tolerance (ABFT). The authors experimentally determined safe undervolting boundaries and developed an error model dependent on voltage and matrix size. GreenMM combines undervolting with modified cuBLAS-MM, correcting errors dynamically. Results on NVIDIA GTX 980 show up to 19.8% power reduction and 9% energy efficiency (GFLOPS/Watt) improvement with only 1.5% performance overhead.

GreenMM limitations include limited compatibility (only via cuBLAS interface), testing only for matrix multiplication (cuBLAS-MM), requiring GPUs with NVML voltage control, no VM or non-Linux OS support, and prerequisite profiling for minimum safe voltage and error models.

Summarizing reviewed solutions, we conclude that the developed optimal power consumption system for GPU sensor data collection should differ. Existing solutions typically target Linux and matrix multiplication. As an alternative, we can develop a Windows system comprising separate modules (testing, data collection etc.), allowing test type customization based on user requests. Additionally, collected data and models from other devices with similar GPUs could be utilized. System modules could run on separate computers interacting as a distributed network.

3. Preliminaries

GPU sensor data collection can be implemented through various methods. We examine GPU-Z, NVAPI (and its PyNVML variant), and Pynvraw.

GPU-Z is free software by TechPowerUp for detailed GPU monitoring and information, Windows-only. GPU-Z provides comprehensive graphics card data (temperature, power consumption etc.) [16], supporting nearly all graphics cards.

GPU-Z's main advantage is complete GPU information collection and logging. However, closed-source code and lack of API access are drawbacks, allowing data retrieval only via processing output CSV files containing rows and columns. Reading these files makes real-time processing in custom software challenging.

NVAPI (NVIDIA Management Library) is a low-level C API provided by NVIDIA for interacting with their GPUs, Windows-only. It enables access to advanced GPU features and settings unavailable through standard drivers or APIs like OpenGL/DirectX. Unlike high-level APIs abstracting hardware details, NVAPI provides direct hardware access for fine-tuning performance and functionality [17].

NVAPI allows obtaining system information including GPU identification, driver versions, and available memory. Beyond data collection, NVAPI manages various GPU aspects like clock speeds and power management.

PyNVML is a Python wrapper for NVIDIA Management Library (NVML) providing an interface for NVIDIA GPU interaction. This library enables monitoring GPU status and characteristics like utilization, memory usage, frequency, temperature, fan speed, and more. PyNVML simplifies GPU monitoring integration into Python applications via its API for load optimization and performance improvement [18]. The library supports both sensor data collection and GPU parameter adjustment (e.g., via *nvmlDeviceSetPowerManagementLimit()*).

Pynvraw is a Python package interfacing with NVAPI for low-level NVIDIA GPU access. Pynvraw monitors and controls GPUs, accessing parameters like current temperature, clock speed, voltage, and workload [19], useful for real-time performance analysis. For our system, using Pynvraw methods (e.g., *api.get_core_voltage()*) instead of some PyNVML methods achieves higher precision comparable to GPU-Z data.

Among reviewed tools, PyNVML and Pynvraw are optimal for integration into our solution. They suit Python Windows implementation well, though limited to NVIDIA GPUs.

4. Implementation

For NVIDIA GPU optimal power consumption modeling, we propose a system of several interacting subsystems. Fig. 1 illustrates component relationships and data flow, with arrows indicating command requests or data retrieval between subsystems.

The system combines custom developments with connected systems/libraries/utilities (e.g., for GPU sensor data reading via API). Each component can be replaced thanks to interface usage (e.g., substituting MSI Kombustor with FurMark). Each system stage can run independently. The data collection and testing stage involves GPU sensor data collection, GPU testing, undervolting systems, and required connected components (GPU APIs etc.) with their interactions. Data collection and verification occur across various operation modes – when reaching extreme values (minimum consumption or maximum GPU/memory frequency) or system instability, more stable parameters are restored and the cycle continues until all parameter combinations are tested.

The Data Acquisition System handles GPU sensor data collection via low-level libraries, outputting string representations and storing in a connected database. The database can be remote depending on connection interface commands. The write method allows direct database storage without data resending. The system also verifies GPU availability and accessibility.

The GPU Testing System runs benchmarks with configurable parameters including test duration and load type, while monitoring GPU state via the Data Acquisition System. The GPU Undervolting System dynamically adjusts GPU parameters (frequencies, power) via low-level APIs, validating changes and recording results in the database. The Data Analysis System applies various methods to evaluate setting impacts on performance (FPS) and energy efficiency (FPS/W). Module interaction uses network sockets with formatted requests including service availability checks and error handling.

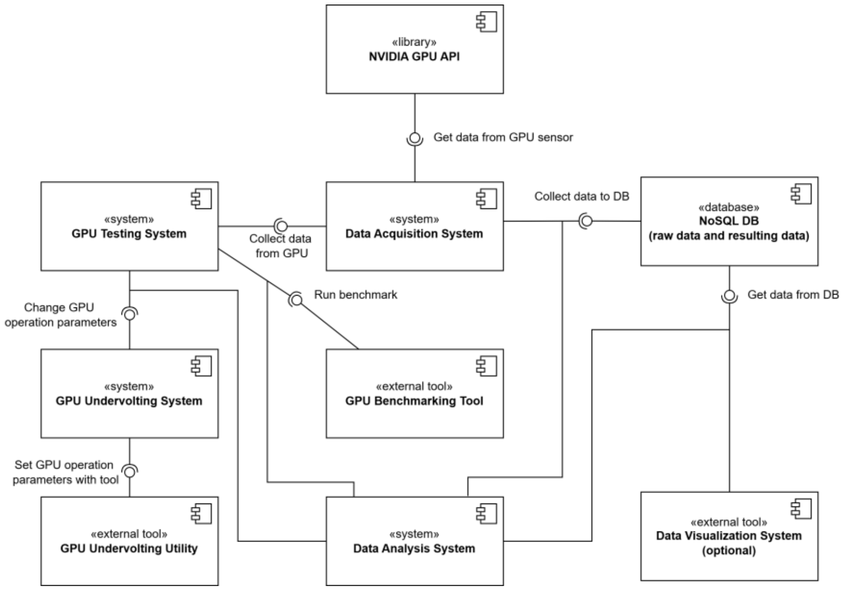

Fig. 1. Component diagram of the developed system.

The data analysis stage involves the operation of a dedicated analysis system that processes data recorded in the database during the previous stage. This system evaluates the correlation between GPU performance metrics and constructs a power consumption model to identify optimal GPU operating parameters. The model enables subsequent retesting and undervolting procedures to compare standard operating conditions with optimized configurations.

Our current Windows implementation collects NVIDIA GPU data via PyNVML and Pynvraw, adjusts GPU parameters via PyNVML and NVIDIA Inspector, runs various MSI Kombustor tests, and stores raw data in MongoDB. Below are example system execution results showing power consumption, FPS, and efficiency graphs from Metabase. "Efficiency" is a custom metric defined in (1), where F is Frames Per Second, P is Board Power Draw (W). "Efficiency" evaluates undervolting effectiveness regarding performance impact – higher values indicate better efficiency.

$$E = \frac{F}{P} \tag{1}$$

Fig. 2, 3, 4 show graphs for NVIDIA GTX 1650 data from glfurrytorus benchmark with +100 MHz GPU clock and +200 MHz memory clock offsets, reducing PL from 70W to 45W in 5W steps. Data are collected several seconds before benchmark start, during operation (FPS and efficiency values only available here), and several seconds after. This is a subset from the collected dataset – testing typically includes multiple benchmark types with varying GPU/memory clock offsets. Clock offsets increase until GPU stability is maintained, controlled via PyNVML exceptions and MSI Kombustor log checks.

As Fig. 2, 3, 4 show, reducing PL improves efficiency while lowering power consumption, but determining optimal stable values requires additional analysis.

One approach for evaluating GPU mode changes (PL and GPU/memory clock offsets) on efficiency (benchmark FPS) is computing correlation coefficients. Values near 1 indicate positive correlation between variables (FPS and GPU parameters). Pearson coefficient measures linear dependence, Kendall and Spearman coefficients measure rank correlation robust to nonlinear/asymmetric data distributions [20].

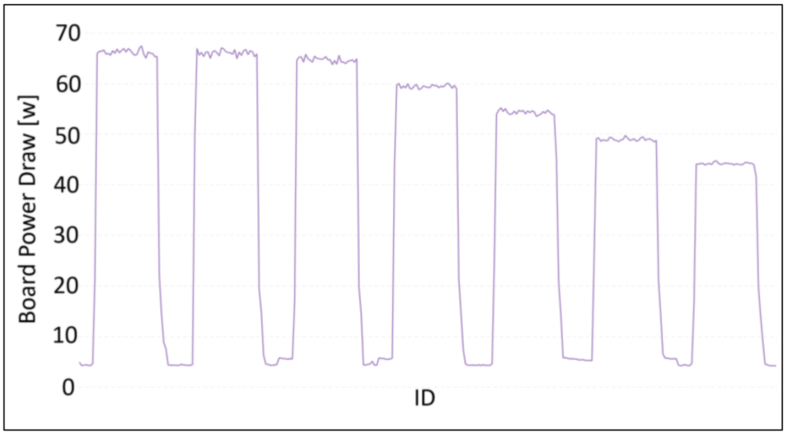

Fig. 2. Effect of Power Limit (75→45 W, 5 W Steps) on Board Power Draw (NVIDIA GTX 1650, glfurrytorus benchmark, GPU +100 MHz, memory +200 MHz).

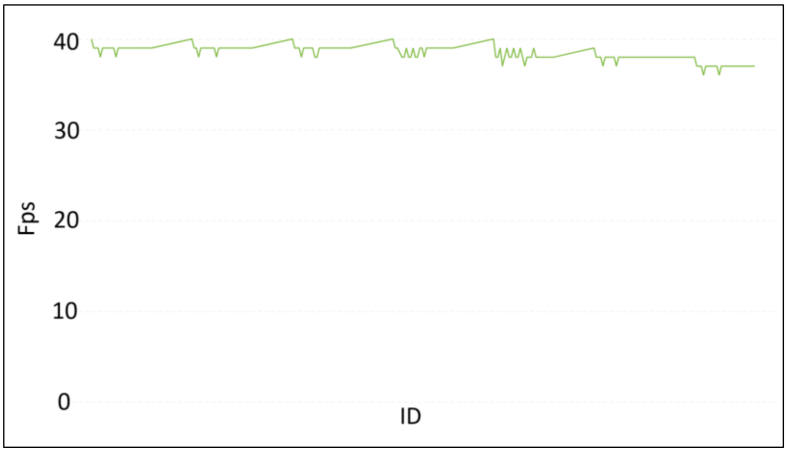

Fig. 3. Effect of Power Limit (75→45 W, 5 W Steps) on FPS (NVIDIA GTX 1650, glfurrytorus benchmark, GPU +100 MHz, memory +200 MHz).

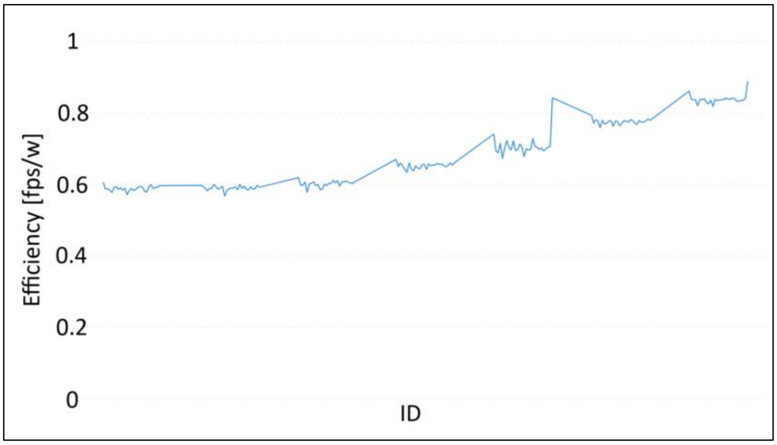

Fig. 4. Effect of Power Limit (75→45 W, 5 W Steps) on Efficiency [FPS/W] (NVIDIA GTX 1650, glfurrytorus benchmark, GPU +100 MHz, memory +200 MHz).

$$r_{xy} = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n\sum x_i^2 - (\sum x_i)^2][n\sum y_i^2 - (\sum y_i)^2]}} \quad (2)$$

$$\tau = \frac{2}{n(n-1)}\sum_{i<j} sgn(x_i - x_j)sgn(y_i - y_j) \quad (3)$$

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \quad (4)$$

Pearson correlation (2) computes linear dependence between variables: x_i, y_i are compared values; n is observations count. Pearson coefficient near 1 between GPU parameter and FPS indicates strong direct linear dependence – GPU parameter increase raises FPS.

Kendall correlation (3) evaluates order consistency: sgn is the sign function (-1, 0, or 1); n is the observations count. This method counts concordant/discordant data pairs. Kendall coefficient helps precisely determine relationship direction and strength.

Spearman correlation (4) uses value ranks: d_i are rank differences; n is the observations count. Spearman coefficient between GPU parameter and FPS may reveal monotonic nonlinear relationships. Spearman helps identify stable trends between FPS and parameters when linear methods fail [21].

Table 1 shows correlation coefficients computed from 1,113,204 MongoDB documents (650,646 with FPS values). Data was collected across test types with varying PLs, GPU clock offsets, and memory clock offsets.

Table 1. Correlation coefficients for FPS performance.

Benchmark type	**Power Limit [W]**	**GPU Clock [MHz]**	**Memory Clock [MHz]**
Pearson Correlation			
Glfurrytorus	0.3757	0.1177	0.8230
Glfurrymsi	0.4352	0.1710	0.8095
glmsi01	0.7438	0.4665	0.0297
glmsi02gpumedium	0.2367	0.3226	0.1645
glphongdonut	0.4317	0.3927	0.5140
Glpbrdonut	0.2524	0.2171	0.7758
gltessyspherex32	0.7222	0.5988	0.0182
Kendall Correlation			
Glfurrytorus	0.2866	0.0822	0.7079
Glfurrymsi	0.3267	0.1179	0.6765
glmsi01	0.6377	0.3686	0.0228
glmsi02gpumedium	0.1717	0.2343	0.1122
glphongdonut	0.3243	0.2830	0.3833
Glpbrdonut	0.1818	0.1502	0.6058
gltessyspherex32	0.5656	0.4656	0.0129
Spearman Correlation			
Glfurrytorus	0.3636	0.1095	0.8248
Glfurrymsi	0.4253	0.1608	0.8105
glmsi01	0.7466	0.4654	0.0298
glmsi02gpumedium	0.2385	0.3315	0.1621
glphongdonut	0.4387	0.3960	0.5315
Glpbrdonut	0.2495	0.2112	0.7865
gltessyspherex32	0.7147	0.6000	0.0183

The analysis reveals that different benchmark tests exhibit distinct FPS dependence patterns on various GPU parameters. The glfurrytorus test demonstrates strong FPS correlation with memory clock offset (Pearson 0.82) with minimal other parameter influence, while glfurrymsi shows pronounced memory frequency dependence (Pearson 0.81) and moderate Power Limit impact. In contrast, glmsi01 reveals maximum FPS correlation with Power Limit (Pearson 0.74) with negligible memory frequency effects, whereas glmsi02gpumedium displays weak correlations with all parameters (coefficients < 0.33) due to its PhysX workload characteristics. The glphongdonut test presents balanced parameter influences (coefficients 0.39-0.53), while glpbrdonut exhibits clear memory clock dependence (Pearson 0.78) with minor other factors. The gltessyspherex32 test shows strong Power Limit (Pearson 0.72) and GPU clock (0.60) correlations with minimal memory impact. These findings clearly show that different benchmark tests require different optimal GPU parameters, and the final parameter values inevitably represent a compromise between performance requirements across various workloads. Graphical summary of the discussed tests is presented in Fig.5.

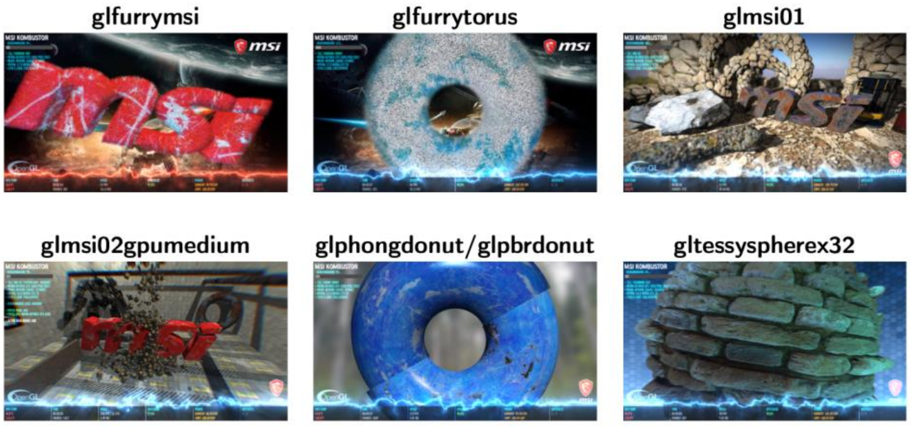

Fig. 5. Tests used for benchmarking (MSI Kombustor).

To obtain optimal GPU parameters, it is necessary to build its model. The process of constructing an optimal GPU power consumption model begins with data preprocessing. The data is converted into a DataFrame, and entries with missing FPS values are removed. Numerical features are normalized, and 15% noise is added to the target variable (FPS or another performance metric) to improve the model's generalization capability. The categorical feature "benchmark test type" is encoded using LabelEncoder, enabling the model to work correctly with this data type.

After preprocessing, the data is split into training and test sets while preserving the proportions of test types. A LightGBM gradient boosting model is configured with parameters optimized for the regression task. These parameters ensure a balance between accuracy and training speed. The trained model is evaluated using the R^2 and MAE metrics, followed by an analysis of feature importance through visualization, which helps understand the influence of various GPU parameters on performance.

The final stage involves optimizing GPU parameters (particularly focusing on power limit, GPU frequency, and memory frequency, as their values can be adjusted via the undervolting system). The identified optimal values are transformed back into the original range and saved to a file for later use. The model's results, including quality metrics and optimal parameters, are displayed for analysis and visualized in interactive charts for easy comparison of GPU performance under different settings.

Gradient boosting is a machine learning method based on ensemble learning, where several weak models (typically decision trees) are combined into one strong model [22]. LightGBM was chosen as the primary method due to its high efficiency and speed when processing large volumes of data.

The optimizer's algorithm is used to predict the target performance parameter for different GPU settings. In general, the target variable is a GPU performance metric, which could be operations per second (FLOPS or TOPS), hash rate (mining speed), etc. In this specific case, the performance metric is FPS (frames per second) in a benchmark. A set of parameters for the current trial is generated within a normalized range.

A complete set of parameters is created and fed into the LightGBM model to predict FPS for different test types, followed by averaging the values. This averaging ensures that the optimal parameters perform well under various GPU workloads. The final step of the method is computing the objective function (5).

Bayesian optimization was chosen due to its efficiency with large datasets. The algorithm optimizes parameters in a relatively small number of iterations [23]. After each iteration, a method is called to evaluate the new set of parameters.

$$f(\theta) = \frac{\overline{FPS} \cdot (1-\alpha)}{P \cdot \alpha + \varepsilon} \tag{5}$$

$$\overline{FPS} = \frac{1}{N}\sum_{i=1}^{N} \tag{6}$$

The objective function (5) represents the ratio of average FPS to power P, where α=0.5 serves as a balancing coefficient between performance and power consumption. The term $\varepsilon=10^{-9}$ is a small constant included to ensure numerical stability during division operations. In this formula, θ denotes the set of optimized GPU parameters, excluding the benchmark test type. The FPS values are averaged across all test types in equation (6).

The fundamental purpose of this formula is to establish an optimal compromise between performance (numerator) and energy efficiency (denominator). The numerator increases the objective function's value as average performance improves, while the denominator decreases it when power consumption rises. The coefficient α enables precise balancing between these two factors: when α=0 the optimization considers only FPS, when α=1 it considers only power consumption, and the value of 0.5 provides equal consideration to both parameters.

The collected raw data consisted of 1,113,204 documents in a MongoDB database, with 650,646 documents containing valid FPS values for the NVIDIA GTX 1650 GPU. This dataset enabled training a LightGBM model with high accuracy, achieving the following metrics: R^2=0.947 and MAE=18.8 FPS.

Fig. 6 presents the feature importance analysis showing the relative influence of different parameters on FPS values. The memory controller load (in %) demonstrates the strongest correlation with FPS, followed by memory clock offset and GPU power consumption. Memory utilization and GPU temperature show moderate influence, while the fixed minimum GPU clock frequency proved least significant as it remained constant during testing. These results clearly indicate that the benchmark tests placed substantial load on the memory subsystem.

The model execution and subsequent prediction yielded optimal parameter values as follows: power limit = 45 W, GPU clock offset = 193 MHz, and memory clock offset = 468 MHz. Fig. 7 presents a comparative analysis between tests conducted with these optimized parameters and default configuration tests. The default configuration (denoted in blue) represents standard settings with power limit = 75 W and zero clock offsets, while the orange markers indicate standard settings with minimum power limit = 45 W.

The experimental results demonstrate a significant performance improvement in terms of FPS metrics, showing an increase of +3.3% compared to the standard test configuration and +8.3% relative to the standard test with minimum power consumption. The power consumption reduction closely matches that of the standard test with power limit = 45 W (representing a 31% decrease), while exhibiting only a marginal 0.1% increase in power consumption that is effectively offset by the corresponding performance enhancement.

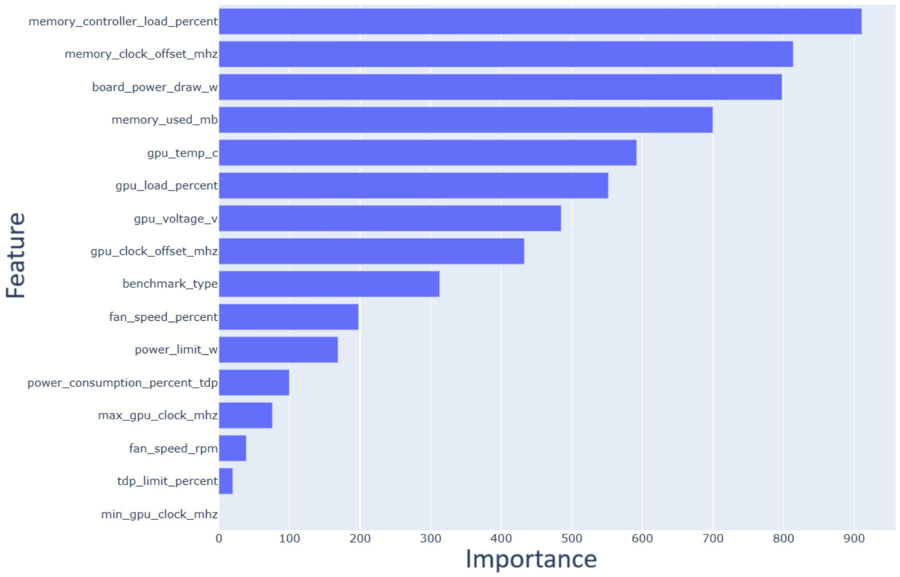

Fig. 6. Model features importance.

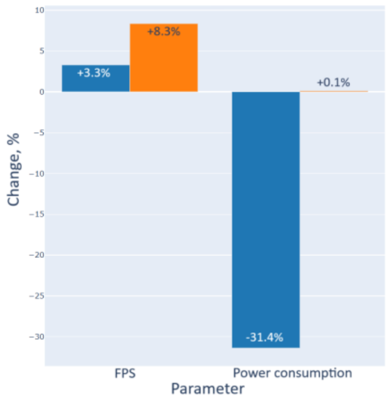

Fig. 7. Performance comparison between optimized and default parameters on GTX 1650.

5. Conclusion

Within reviewed GPU data collection tools, we examined GPU operation parameters and datasets obtainable via software-defined counters. We studied methods for obtaining graphics processor information under load. We compared existing modeling systems to determine an optimal scheme for our solution.

Current software is Python-based, we made it available in GutHub [24]. The resulting dataset, including all benchmark results and configuration parameters, is also available in [24]. We developed a modular system collecting GPU sensor data into a database, running benchmark stress test cycles, and gathering extensive information through GPU parameter adjustments via undervolting. Distributed component operation enables workload distribution across devices (e.g., data collection on one device, analysis on another). Our system performs comprehensive data

analysis to identify correlations and patterns, builds optimal power consumption models using LightGBM, and predicts optimal GPU operating parameters through Bayesian optimization methods. The obtained correlation coefficients revealed distinct optimal parameter configurations for different workload types, with final values representing a performance compromise between various workload types. The optimized parameters enabled an 8.3% FPS improvement for the NVIDIA GTX 1650 while maintaining minimal power consumption (45 W).

References

[1]. Falk H. Personal computer graphic cards. The Electronic Library, vol. 12, no. 2, pp. 127–130, 1994.
[2]. Fatahalian K., Houston M. A closer look at GPUs. Communications of the ACM, vol. 51, no. 10, pp. 50-57, 2008.
[3]. Garland M, Kirk D. B. Understanding throughput-oriented architectures. Communications of the ACM, vol. 53, no. 11, pp. 58-66, 2010.
[4]. Gorlatch S., Garanina N, Staroletov S. Using the SPIN model checker for auto-tuning high-performance programs. Journal of Mathematical Sciences, pp. 1-13, 2025.
[5]. Кондратьев Д.А., Старолетов С.М., Шошмина И.В., Красненкова А.В., Зиборов К.В., Шилов Н.В., Гаранина Н.О., Черганов Т.Ю. Соревнования по формальной верификации VeHa-2024: накопленный в течение двух лет опыт и перспективы. Труды ИСП РАН, том 37, вып. 1, 2025 г., стр. 159-184. DOI: 10.15514/ISPRAS–2025–37(1)–10. / Kondratyev D.A., Staroletov S.M., Shoshmina I.V., Krasnenkova A.V., Ziborov K.V., Shilov N.V., Garanina N.O., Cherganov T.Y. VeHa-2024 Formal Verification Contest: Two Years of Experience and Prospects (in Russian). Trudy ISP RAN/Proc. ISP RAS, vol 37, issue. 1, pp. 159-184, 2025.
[6]. Muralidhar R., Borovica-Gajic R, Buyya R. Energy efficient computing systems: Architectures, abstractions and modeling to techniques and standards. ACM Computing Surveys (CSUR), vol. 54, no. 11s, pp. 1–37, 2022.
[7]. Pei Q., Yuan Y., Hu H., Wang L., Zhang D., Yan B., Yu C., Liu F. Working Smarter Not Harder: Hybrid Cooling for Deep Learning in Edge Datacenters. IEEE Transactions on Sustainable Computing, 2025.
[8]. Murdock K., Oswald D., Garcia F. D., Van Bulck J., Piessens F., Gruss D. Plundervolt: How a little bit of undervolting can create a lot of trouble. IEEE Security & Privacy, vol. 18, no. 5, pp. 28–37, 2020.
[9]. Colmant M., Rouvoy R., Kurpicz M., Sobe A., Felber P., Seinturier L. The next 700 CPU power models. Journal of Systems and Software, vol. 144, pp. 382–396, 2018.
[10]. Fieni G., Acero D.R., Rust P., Rouvoy R. PowerAPI: A Python framework for building software-defined power meters. Journal of Open Source Software, vol. 9, no. 98, p. 6670, 2024.
[11]. Savikov D., Shintyapin I., Staroletov S. Concept of building power models for central processors using the PowerAPI toolkit (in Russian). In Modern digital technologies: proceedings of the III All-Russian scientific-practical conference (03 June 2024). Barnaul: AltSTU, pp. 211–215, 2024. Available: https://elibrary.ru/item.asp?id=68573384.
[12]. Gulin A., Filimontsev I., Staroletov S. Analysis of data from central processors for building their power models using the PowerAPI toolkit (in Russian). In Modern digital technologies: proceedings of the III All-Russian scientific-practical conference (03 June 2024). Barnaul: AltSTU, pp. 138–142, 2024. Available: https://elibrary.ru/item.asp?id=68573361
[13]. Group S.R. et al. PyJoules: a Python library to capture the energy consumption of code snippets. University of Lille and Inria, 2021.
[14]. Zamani H., Tripathy D., Bhuyan L., Chen Z. SAOU: Safe adaptive overclocking and undervolting for energy-efficient GPU computing. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, pp. 205–210, 2020.
[15]. Zamani H., Liu Y., Tripathy D., Bhuyan L, Chen Z. GreenMM: energy efficient GPU matrix multiplication through undervolting. In Proceedings of the ACM International Conference on Supercomputing, pp. 308–318, 2019.
[16]. Dong Z., Wang Z., Yuan J. Research and application of GPU pass-through based on KVM in graphics workstation. In 2018 3rd International Workshop on Materials Engineering and Computer Sciences (IWMECS 2018). Atlantis Press, pp. 354–357, 2018.
[17]. Czarnul P., Proficz J., Krzywaniak A. Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments. Scientific Programming, vol. 2019, no. 1, p. 8348791, 2019.

[18]. Torres L.A., Barrios C.J., Denneulin Y. Computational resource consumption in convolutional neural network training–a focus on memory. Supercomputing Frontiers and Innovations, vol. 8, no. 1, pp. 45–61, 2021.
[19]. pynvraw, 2021. Available: https://github.com/JustAMan/pynvraw, accessed Jul. 15, 2025.
[20]. Shaqiri M., Iljazi T., Kamberi L., Ramani-Halili R. Differences Between The Correlation Coefficients Pearson, Kendall And Spearman. Journal of Natural Sciences and Mathematics of UT, vol. 8, no. 15-16, pp. 392–397, 2023.
[21]. De Winter J.C., Gosling S.D., Potter J. Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. Psychological methods, vol. 21, no. 3, p. 273, 2016.
[22]. Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems, vol. 30, 2017.
[23]. Frazier P.I. Bayesian optimization. In Recent advances in optimization and modeling of contemporary problems. Informs, pp. 255–278, 2018.
[24]. GPU Power Model, 2025. Available: https://github.com/GulinAlexey/GPU_Power_Model, accessed Jul. 15, 2025.

Информация об авторах / Information about authors

Алексей Николаевич ГУЛИН – магистрант кафедры прикладной математики АлтГТУ по направлению “Программная инженерия”. Сфера научных интересов: энергоэффективные вычисления, оптимизация вычислительных систем.

Alexey Nikolaevich GULIN – master student at the department of Applied Mathematics, Software Engineering direction, AltSTU. Research interests: energy-efficient computing, computational system optimization.

Сергей Михайлович СТАРОЛЕТОВ – кандидат физико-математических наук, доцент. Сфера научных интересов: формальная верификация, model checking, киберфизические системы, операционные системы.

Sergey Mikhailovich STAROLETOV – Cand. Sci. (Phys.-Math.), associate professor. Research interests: formal verification, model checking, cyber-physical systems, operating systems.