ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC [004.2+004.75]:504.062:620.9:519.216.3 **doi: 10.32620/reks.2024.2.10**

Oleksandr MAMCHYCH, Maksym VOLK

Kharkiv National University of Radioelectronics, Kharkiv, Ukraine

A UNIFIED MODEL AND METHOD FOR FORECASTING ENERGY CONSUMPTION IN DISTRIBUTED COMPUTING SYSTEMS BASED ON STATIONARY AND MOBILE DEVICES

*The **subject of research** in this article is the forecasting of energy consumption when computing distributed tasks on computer networks built on the basis of server solutions and distributed systems based on personal smartphones. The **goal** of this study was to create a universal computing energy cost prediction model that can be applied to both traditional and mobile cloud systems. **Tasks**: conduct an analysis of energy-saving approaches and technologies used to calculate data; consider computer system models and actions with them, namely: model of distributed job, model of distributed computing system, model of distribution strategy; develop a common and uniform dynamic method of forecasting spent energy with a focus on heterogeneous systems; conduct a study of the proposed approach on stationary and mobile devices. **The obtained results include.** The results of the experimental measurement of the energy consumption of mobile digital systems and stationary ones are presented. The energy efficiency of computing on GPUs of a stationary device based on CUDA technology and GPUs on mobile devices based on Apple Metal technology was determined. Computation during the calculation of 600 frames on a distributed system from mobile devices with failure settings showed a consumption of 15320 joules of energy. Simulation of computing on a distributed system with stationary devices showed a consumption of 52806 joules of energy. This gives us 3,45 times the consumption benefit from computing on mobile devices. Forecasted consumption is also very accurate. **Conclusions.** The energy consumption assessment model proved to be quite effective. The results of the experiments show that the energy consumption estimation model takes into account the features of the hardware platform where data processing is performed. Computation of data on the GPU of stationary devices loses energy efficiency to a similar implementation on the GPU of Apple Metal from mobile devices. Therefore, the presented results demonstrate the rationality of using mobile graphics processors for energy-efficient information processing.*

Keywords: *graphics processor; energy efficiency; distributed system; cloud computing; green computing; model; mobile device; multithreading.*

1. Introduction

1.1. Motivation

Information technologies implement the functions of monitoring, information processing, and management, which directly affect the technical level of energy systems for both traditional and green energy. Without information technologies, it is impossible to effectively respond to challenges in the field of conservation and increase energy resources. However, every year, information technologies in this context acquire an independent role as IT systems become more and more active consumers of energy [1, 2]. According to various sources, they account for up to 3% of the total amount of energy consumed.

Various calculation tasks are integral components of information systems. The increasing complexity of these tasks increases the demand for computing power [3, 4]. If the task cannot be computed using local resources, it is usually computed in large data centers on specialized equipment. This, in turn, leads to an increase in the demand for cloud computing power. To expand the available cloud computing capacities, it is necessary to manufacture new hardware and create new data centers.

1.2. State of the art

At the level of energy savings in electronic components, researchers distinguish between two groups: hardware (technical) and software. They allow for detail in the context of information technology.

Hardware [5, 6]. For hardware, primarily microcircuits of various purposes and designs, there are solutions that reduce energy consumption. They are based on the application of special electronic circuits, technologies, and computing paradigms; modes of reduced energy consumption ("sleeping" and "semi-sleeping") for the entire crystal or its individual parts; schematic design solutions that minimize the number of simultaneously switching crystal elements and thus reduce current surges; adaptation schemes in redundant structures, the channels of which operate at the limit

reduced voltage (in this case, failures caused by unstable operation are possible); and special settings for programmable crystals, etc. Similar approaches are used in more complex hardware components (modules, channels), where various combinations of methods can be used [7, 8].

Software tools [9, 10]. The approaches used for software tools are not as clear-cut as their impact on energy savings is indirect. However, the concept of "green software" [11, 12] has been established in the scientific and technical lexicon, based on the understanding that: each operator, language construct and program module can have its own energy metric (that is, it is characterized by the energy consumed by the platform on which the program is implemented); different software solutions vary in the amount of resources used (not only energy) and can be optimized (including according to the energy criterion); since energy consumption or energy efficiency become attributes of software quality, it is advisable to use a process approach for their assessment and provision; to evaluate energy meters of software options, it is necessary to use special hardware and software solutions that will ensure the accuracy of measurements of the energy consumed during the execution of various applications. Thus, for green software, the tasks of clarifying the quality model, which should include characteristics that directly and/or indirectly consider the energy component, defining a set of metrics used, developing methods for measuring relevant characteristics, and optimizing processes and products obtained, are important at different stages of the software life cycle according to the specified criteria.

In absolute terms, this is a colossal amount that can be compared with the energy consumption of the largest European economies. For example, a modern cloud IT infrastructure, including a data center and several thousand computers, consumes 10-15 megawatts daily.

Currently, there are up to 20 billion personal mobile devices in the world, which have a very impressive aggregate computing power [13, 14]. They can also be used to calculate a whole range of problems.

In addition, most existing scientific research works are aimed at maximizing the efficiency of execution of certain algorithms. Usually, the main metrics are the minimization of task execution time and the minimization of idle time of individual parts of computer systems. Very little attention is paid to minimizing energy and traffic costs. Therefore, potential computing networks built from personal smartphones receive very little attention or remain outside the scope of newest approaches.

This work aims to study methods for forecasting energy consumption when computing distributed tasks on computing networks built from smartphones, as well as compare the energy consumption of traditional distributed computing systems based on server solutions and distributed computing systems based on personal smartphones.

The objective of this work is to create a universal model for forecasting energy costs in computing, which can be applied to both a traditional cloud system and a cloud system based on mobile devices. The model must meet the following criteria: the same dynamic algorithm for scheduling tasks, focus on heterogeneous systems, the ability to measure and predict energy costs, take into account the limited reliability of node access to the network and related overheads, and be able to estimate the energy costs of both mobile equipment and conventional computing hardware.

Modern microprocessor manufacturers strive daily not only to increase the performance of their chips but also to reduce their overall energy consumption. Mobile phones (smartphones), tablets, laptops, and other devices that are convenient to carry and use in everyday life have become integral attributes of our time. To increase energy efficiency, manufacturers use various circuit-technical and software solutions. One straightforward method to extend a device's lifespan is by increasing the battery capacity [15]. Further, there are methods of reducing energy consumption by the periphery, namely special low-power displays, low-voltage microcircuits in communication and navigation modules, and low-voltage processors [16]. The next stage is the special operating modes of processor devices, which allow the device to be put into standby, sleep or deep sleep mode, with the possibility of quick awakening, but at the same time power consumption is reduced from hundreds of mA to tens of μA [17]. The final stage is energy-saving software [18]. There are many programs that help save the resources of the already voracious computing core of any microprocessor device. Such programs have gained popularity since the days of computers and are now actively used by mobile devices.

The paper [19] presents a comprehensive study on forecasting energy consumption in electrical buildings using smart grid technologies, focusing on the application of artificial neural networks and k-nearest neighbor algorithms. This study emphasizes green computing by optimizing the computational efficiency of the GPU processing unit during the forecasting process. This paper outlines an approach to outlier detection and correction, which significantly enhances the quality of the dataset used for training forecasting models.

The paper [20] provides an insightful exploration of the use of Particle Swarm Optimization (PSO) for enhancing energy efficiency in smart home environments. This study addresses the critical issue of high energy consumption in smart homes by developing

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

a mobile application that employs PSO for task scheduling of smart home appliances.

One notable study describes methodologies related to RAM energy consumption. The authors propose to transform the program code in such a way as to organize work with smaller memory areas and with less energy consumption for each memory access operation. In another study [21], instruction-level models with high clock accuracy were used to evaluate the complex impact of software and hardware optimization on energy efficiency (the processor operation was simulated before each operating cycle). However, despite all previous work, the energy consumption of a specific program has always been estimated using a specific target architecture.

Many studies have investigated the advantages of certain approaches to planning the distribution of tasks on a distributed system. However, the primary metric remains the minimization of computational task execution time. Any model suffers from distribution overhead to one degree or another, so such methods are forced to duplicate part of the calculations or transfer a lot of data many times, which leads to a decrease in the energy efficiency of the task even compared to the execution on a single computing device. In addition, task planning models for distributed systems usually expect fairly large bandwidth and fast response of network interfaces, but personal mobile devices have limited bandwidth and long response times. All such works cannot automatically be applied to a distributed system from mobile devices; any such system will be purely heterogeneous [22].

There are many studies devoted to cloud computing [23, 24] that assess energy efficiency calculations and energy consumption forecasting in cloud computing. However, they always work with server equipment traditional for data centers, almost not considering data exchange and consumption of peripheral devices [25, 26]. These are usually limited to the telemetry data of the computing devices themselves (CPUs and GPUs), while peripherals and network infrastructure can also consume significant amounts of power. When calculating energy costs on a smartphone, it is much easier to consider the consumption of the device as a whole, together with the costs of data transmission and peripherals.

Some studies describe the behavior of computing systems with limited reliability, but they do not study the effect of limited reliability of a node on energy consumption [27].

Studies have shown that mobile computing systems themselves are more energy-efficient than traditional data center equipment [28].

To evaluate the energy efficiency of the computing cloud from mobile devices, it should be considered as a purely heterogeneous system with limited reliability of nodes, limited bandwidth of network interfaces, and exclusively dynamic planning. There is no work that looks at the cloud from mobile devices from this point of view and compares it with traditional clouds based on data centers on the same tasks and considers the maximum total energy costs.

1.3. Features of computing and energy efficiency of CPU and GPU

Computing equipment has long ceased to be a simple device. Today, tasks solved with its help represent the modeling of processes occurring in reality. Therefore, the natural parallelism of the real world flows into the formulation of the problem and the corresponding mathematical model. The extent to which the architecture of computing devices is consistent with the properties of the real world determines how effective the solution of the tasks will be. In recent years, it has become clearly noticeable that the evolution of the CPU has lowed due to many technological limitations. Manufacturers can no longer significantly increase the operating frequencies of the CPU, and a simple increase in the number of processor cores while preserving the original architecture (Table 1) does not provide the desired increase in performance when solving real problems. Additionally, the complexity of a single core growth to achieve higher performance per tick at the cost of higher energy consumption. At the same time, exponential growth is observed in GPU performance and in its architecture, which reflects the properties of the real world. Initially, the final result of graphic data processing on the GPU was a set of pixel colors. The computation of pixel colors perfectly fits an array of simple processors performing similar computations. Therefore, GPUs are mainly intended for work with large amounts of similar data.

Focusing on the processing of many independent pixels, the GPU was originally built as a parallel matrix calculator. With the expansion of the set of graphic operations in the graphic core, pipeline parallelism was added to matrix parallelism. These properties of the architecture, originally laid down in the GPU, also determined other features, such as the command system and thread management.

The CUDA computing architecture is based on the concept of Single Instruction Multiple Data (SIMD) and the concept of a multiprocessor. The SIMD concept assumes that one instruction can simultaneously process a large amount of data. A multiprocessor is a multi-core SIMD processor that allows only one instruction to be executed on all its cores at any given time.

This architectural solution allows calculations to be performed on the GPU more efficiently than on the central processor, provided that the task can be divided into multiple threads. Similar to running on a CPU, a

bottleneck in a computing system is memory access. On GPUs, memory is usually accessed through a wide memory bus, which supports overwhelming resulting throughput if the data is properly organized. The result performance is great, yet it highly depends on the smooth and predictable loading of multiple cores. This is one of the main principles of CUDA, which can significantly improve the performance of the system as a whole. Another opportunity to improve performance is to combine several global memory requests into one, called a transaction or coalescing global memory access. To combine requests into a transaction, several conditions must be met. First, concurrently executing threads must access either 32-bit words, resulting in one 64-byte block, or 64-bit words, resulting in one 128-byte block. If 128-bit word access is used, the result will be two transactions, each returning 128 bytes of information.

Table 1

Comparison of the characteristics of CPU and GPU

Parameter	**CPU**	**GPU**
Architecture	Serial architecture with added vector instructions	Initially parallel architecture
Command system	SISD	SIMD
ALU	Complex	Simple
Solving the memory access problem	Large cache memory, branch prediction	Wide bus, simplified prediction, batch access
Number of threads	1-2 per core, up to dozens of cores per chip	Thousands of cores per multiprocessor
Data format	All formats: integers, reals of various precision	Mostly single and half precision

Second, threads must access memory locations sequentially, each subsequent thread must be assigned the next word in memory, and all words must be within the memory block being accessed. In software implementations based on CUDA technology, data must be structured in such a way that it is almost always possible to combine it into transactions when accessing global memory, because otherwise the performance of calculations on the GPU is sharply reduced and may even be lower than sequential data processing on the central processor. The primary goal of moving calculations to the GPU is usually to obtain significant performance gains. However, the GPU architecture has another important property - high energy efficiency for many high-paralleled tasks [29]. The main prerequisites for this are two factors: most of the area of the GPU crystal is allocated to ALUs, which are directly involved in data processing, and not in the tasks of controlling the sequence of commands; Due to the massive parallelism of calculations, it becomes possible to reduce the frequency and, as a result, energy consumption while maintaining high performance.

1.4. Objectives and the approach

We can distinguish 2 major objectives in this article:

1. Develop a model that is able to evaluate energy consumption on distributed systems based on conventional servers and mobile devices during the execution of similar useful payloads.

2. Comparison of energy consumption of 2 systems: based on conventional server equipment (GPU) and mobile devices (mobile SoC)

The approach is based on identical evaluation tasks having similar algorithms and implementations and being run on different distributed systems. In addition, the approach takes into account not only the energy consumed by the computation unit itself but also the energy consumed during data transferring; since this is a very important aspect of computing systems based on personal mobile devices.

The structure of this article is as follows:

1. Description of a model of distributed computing as a union of 3 models: a distributed computing system, a distributed job, and a distribution strategy model (subsections 2.1-2.4).

2. Initialization and running of the distributed computing model with estimated parameters (subsections 2.5-2.6).

3. Benchmarking of real hardware to estimate energy consumption during computing. To run the model, we need to set up the hardware parameters, which cannot be taken from public sources; hence, we need to collect them (subsection 3.1).

4. Running the real computation. Collecting measurements from computing a real job on real hardware: consumed energy and duration (subsection 3.2).

5. Comparison of the model output and real computing run and discussion of the results. The purpose here is to understand the accuracy of the model from an energy consumption perspective (section 4).

6. Summarizing the achieved results from the accuracy and universalism perspective. Suggesting potential ways for future improvements in accuracy, universalism, and security (Conclusion).

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

2. Case study. Development of a model for forecasting energy consumption

2.1. Discrete model of a distributed computing

To simulate a distributed system, we will use a discrete model because of the relative simplicity of its further implementation. To successfully model the energy costs of computing, the model must consider the specifics of the hardware, the structure of the task itself, and the distribution method. Therefore, the general model can be represented as a system of three smaller, relatively independent discrete models: the model of the computing system, the model of the computing job, and the model of the computing strategy. We denote them as follows:

$$M = \langle M_C; M_J; M_S \rangle, \quad (1)$$

where M_C – model of the computing system;

M_J – model of computing job;

M_S – model of computing strategy.

Each model is a set of interrelated constants and variables. For example, a model of a computing system should contain information about the computing resources of all nodes included in the model, as well as information about the state of the node at the moment: online or offline. The job model is a set of constants that describe the complexity of certain operations in the execution process. The computing strategy model contains a set of variables and rules for transitions between states that describe the general state of job on the computing system at each discrete moment in time - a flow. Then the computational energy is a function of these three models:

$$E(M) = (M_C; M_J; M_S). \quad (2)$$

2.2. A model of a distributed computing system

The distributed computing system of a traditional data center may be homogeneous, but the distributed computing system of mobile devices is heterogeneous by default because it consists of devices with different computing powers and communication channel speeds. Furthermore, unlike traditional servers, the typical operation of a mobile device includes periodic losses of network communication, and consequently, connectivity with the computing system's control center. Let's assume that if the node has a network connection with the control center, it is in online mode; otherwise, it is in offline mode. Thus, the simplest model of a single mobile device can be described as computing power (as the number of computing operations per tick), network access speed (as the maximum amount of data that can be transferred per tick), and network access state (online and offline). To describe the state in the network, we use the Markov process: as the probability of changing one state to another during one tick. Since the main interest for this study is energy consumption, for each device it is also necessary to specify the computing device energy consumption (as the amount of energy consumed by a node in computing per tick) and the data transmission energy consumption (as the amount of energy consumed by the device in data transmission per tick). Therefore, a computing device can be defined as follows:

$$N_i = \langle n_i; R_i^{on}; R_i^{off}; p_i; c_i; e_i^p; e_i^c \rangle, \quad (3)$$

where n_i – availability at the moment of time;

R_i^{on} – probability of transition to the online state;

R_i^{off} – probability of transition to the offline state;

p_i – computing power in operations per 1 tick;

c_i – channel bandwidth in bytes per 1 tick;

e_i^p – energy consumption in joules during calculation for 1 tick;

e_i^c – energy consumption in joules during data transfer for 1 tick.

If n_i is equal to 1, then the node is considered online and can perform calculations, if 0, then it is offline and cannot perform calculations. A transition from 0 to 1 has a probability R_i^{on} every tick, a transition from 1 to 0 has a probability R_i^{off}. Therefore, n_i` on the next tick is calculated according to the formula

$$n'_i = \begin{cases} p(R_i^{on}), & n_i = 0; \\ 1 - p(R_i^{off}), & n_i = 1; \end{cases} \quad (4)$$

where $p(R_i^{on})$ and $p(R_i^{off})$ are probabilistic values of uniform distribution.

The other parameters N_i are constant and do not change during the calculation. Then the model of the computing system can be represented as an array of all computing nodes. Let C be the number of nodes in the model. The computer network model can then be defined as follows.

$$M_C = \langle C; N_i, \ i = 0...C-1 \rangle. \quad (5)$$

Or in the expanded view:

$$M_C = \langle C; n_i; R_i^{on}; R_i^{off}; p_i; c_i; e_i^p; e_i^c, i = 0...C-1 \rangle, \quad (6)$$

where C – number of nodes in the model.

2.3. A model of distributed job

The model of distributed computing can be represented as a set of interconnected tasks and intermediate data - artifacts. Let each task require a certain set of artifacts for computation and generate one artifact in the computation process. Let the artifacts used to calculate the problem and generated as a result of the calculation be characterized by the amount of data and denoted by A_i. Let the task be characterized by the number of computational operations that must be performed to obtain its result, which is denoted by Ti. Let the result of the task T_i be the artifact A_i. Computation of artifact Ai by task Ti requires some artifacts to be computed and available. Let such a dependence be given by the matrix D_{ij}: if the calculation of the i-th artifact requires the j-th, then D_{ij} =1, otherwise D_{ij} =0. Let J be the number of artifacts and tasks in the work.

Thus, the computational work can be represented as follows:

$$M_J = \left\langle T_i; A_i; D_{ij},\ i = 0...J-1,\ j = 0...J-1 \right\rangle, \qquad (7)$$

where T_i – computation complexity of a task;

A_i – amount of data contained in an artifact;

D_{ij} – matrix denoting dependencies between tasks and artifacts.

The number of artifacts needed to calculate the i-th artifact is calculated by the following formula:

$$In(M_j, i) = \sum_{j=0}^{J-1} D_{ij}. \qquad (8)$$

The number of artifacts, for the calculation of which it is necessary to have the data of the i-th artifact, is calculated according to the formula:

$$Out(M_j, j) = \sum_{i=0}^{J-1} D_{ij}. \qquad (9)$$

Some artifacts are primary inputs, which are inputs to the work itself. Some artifacts are not used to calculate other artifacts, such artifacts are the raw data of the work. Let the artifacts for which $In(M_J,i)=0$ be called inputs, and the artifacts for which $Out(M_J,j)=0$ be called outputs. Input artifacts do not require computation and are assumed to be known at the beginning of the computation. In addition, T_i is assumed to be 0 for all input artifacts.

2.4. Distribution strategy model

For the task to be completed, it must be assigned to some node. A node must receive a computation instruction, download the required artifacts to perform a task, execute the task, and then transmit the computed artifact to the system, i.e., the control center of the computing network. Thus, the model should describe:

– a list of artifacts available to the system at a moment in time; let such a list be called the scope of the system;

– a list of artifacts available to each node at a point in time; let such a list be called the scope of the node;

– the task assigned to be performed on a node at a moment in time;

– the task that a certain node performs at a given moment in time;

– an artifact that a certain node transmits at a given moment in time;

The scope of all nodes can be described by the matrix s of size CxJ, s_{ij} = 1 if the i-th node has data of the j-th artifact, otherwise - 0. The scope of the system can be described by the array g of dimension J, g_j =1 if the system has data of the j-th artifact, otherwise - 0. Initially, no node possesses any artifact data, and the system contains only data for incoming artifacts, as outlined in Formula (10). In order for an artifact to become available in a scope, this artifact must either be transferred to this scope from another, or calculated from existing artifacts. Transmission is possible only between the system and node, i.e., from node to system or from system to nodes. For simplicity, transmission between nodes within this model is not possible.

Task assignment to a computing node defines the task's relationship with that node. Since each node and task is ordered by index, such a relation can be specified by an array a of size J, where a_i is the destination node number or -1 if the task is not assigned to any node.

The instantaneous status of the task calculation on the node contains the task number and the number of operations that must be performed on the node to perform the assigned task and obtain the corresponding artifact. They can be described as arrays x and x' of size C. Let x_i be the number of the task running on the i-th node, or -1 if no task is currently running on the i-th node. Let x_i' be the number of operations that must be performed to complete the task on the i-th node.

The current data transfer status encompasses the artifact number and the volume of data required to finalize the transfer. They can be described as arrays t and t' of size C. Let t_i be the number of artifact transmitted by the i-th node, or -1 if no transmission occurs. Let t_i' be the amount of data that must be transferred to complete the transfer of the artifact by the i-th node.

The direction of transmission on the i-th node is determined by the presence of this artifact in the system's scope: if the data artifact is in the system's scope, then the transmission goes from the system to the node, if this

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

artifact is not in the system's scope, then the transmission goes from the node to the system.

Therefore, the distribution strategy model can be articulated as a set of the following variables:

$$M_S = \left\langle \begin{matrix} g_j; s_{ij}; a_i; x_i; x'_i; t_i; t'_i, \\ i = 0...C-1,\ j = 0...J-1 \end{matrix} \right\rangle, \quad (10)$$

where g_j – presence of j-th artifact's data in the system scope;

s_{ij} – presence of j-th artifact in the scope of i-th node;

a_i – node number i-th task is assigned to;

x_i – number of a task currently running on i-th node;

x_i' – number of operations to finish xi task;

t_i – number of an artifact being transferred by the i-th node;

t_i' – amount of data to be transferred to finish ti transferring.

Let the number of tasks assigned to node i be calculated using the following formula:

$$\text{Assign}(M_S, i) = \sum_{j=0}^{J} \begin{cases} 0, & a_j \neq i; \\ 1, & a_j = i; \end{cases} \quad (11)$$

The amount of data that needs to be transferred to the i node to start calculating the j subtask:

$$\text{Dl}(M_S, i, j) = \sum_{k=0}^{J-1} D_{jk} \left(\begin{cases} 0, & s_{ij} = 1 \\ 1, & s_{ij} = 0 \end{cases} \right). \quad (12)$$

A task is considered available for distribution when all the artifacts necessary for its calculation are within the scope of the system. The availability of a subtask for distribution is calculated using the following formula:

$$\text{Avail}(M_S, i) = \sum_{j=0}^{J-1} D_{ij} \cdot g_i. \quad (13)$$

The computation of work is considered complete when all artifacts are available within the scope of the system. The number of artifacts available in the system scope is calculated as the sum of all non-zero values in the g array:

$$\sum_{j=0}^{J-1} g_j. \quad (14)$$

Subtask i is considered available for computation on the j-th node when all artifacts required are within the scope of the corresponding node. Availability can be determined by the following formula:

$$\text{Avail}(M_S, i, j) = \sum_{k=0}^{J-1} D_{ik} \cdot g_{ji}. \quad (15)$$

2.5. A model initialization

Models M_J and M_C contain constants other than n_i; therefore, they are assumed to be given from the beginning. Array n is filled with 0 – all nodes are considered offline at the beginning. Next, the Markov process is simulated according to formula (1) for a sufficiently large number of ticks to bring the system to a general state.

Array g is filled as follows:

$$g_i = \begin{cases} 1, & \text{In}(M_j, i) = 0; \\ 0, & \text{In}(M_j, i) = 1. \end{cases} \quad (16)$$

Arrays a, x, x', t, t' and matrix s are filled with 0.

Let E = 0 – the accumulated amount of spent energy, the calculation of which is the goal of this model.

2.6. A method for calculating spent energy

The method for calculating the spent energy in a general form. This method is based on the application of a cyclic algorithm, in which certain steps can have different implementations. This article offers implementations for all steps, but other implementations are possible that better reflect certain practical tasks:

1. Simulation of the Markov process of node availability according to formula (4) for n.
2. Unassigning tasks from offline nodes: for all $i = 0...J-1$ execute: if n_{ai}=0, set a_i = -1.
3. Canceling the assignment of tasks to offline nodes. For all i=0...C-1 execute: if n_i =0, set x_i =-1, set x_i`=0, set $t_i$ =-1, set $t_i$`=0; if n_i=0, for all j=0...J-1 set s_{ij} =0.
4. Execute a naive heuristic task assignment strategy for each node described.
5. Implementation of the local scheduling strategy for each node.
6. Perform data transfer simulation for each node.
7. Running a simulation calculation for each node.
8. Checking the condition of the task according to formula (8): if $\sum_{j=0}^{J-1} g_j = J$, completion.

Implementation of a naive heuristic strategy for subtask assignment to nodes. In this case, it is possible to use different distribution models; however, within the scope of this work, a naive heuristic distribution model is used. This strategy model is based on a greedy algorithm.

At each step, it tries to allocate to each node the task that will require the least energy costs from the point of view of data transmission.

Algorithm for the i-th node:

1. If $n_i = 0$, the node is not reachable, go to the end.
2. If $a_i >= 0$, the node already has an assigned subtask, go to completion.
3. If there is such $j \in [0; J-1]$ a value $Avail(M_S, j)=0$ according to formula (7), go to destination, otherwise - go to completion.
4. Set the value $a_i = j$.
5. Completion.

The same scheduling strategy is executed for each node. It provides the following principles:

1. If no subtask is assigned to a node, the node does not load data from the system or compute, but can perform data load on the system.
2. Data transfer and calculations are performed independently and in parallel.
3. If there is an artifact in the node's scope that is not available in the system's scope, then downloading that artifact is the highest priority and the node tries to download it immediately
4. If the node is allocated a subtask, then the node starts downloading artifacts from the system necessary for its calculation
5. If the artifacts needed to compute the assigned subtask are available in the node's scope, the node starts computing the subtask
6. If a node goes offline, all local transfer and computation progress is lost
7. When a node goes online, its scope is assumed to be empty, and all data must be loaded from the beginning for calculation

Algorithm for the i-th node:

1. If $x_i \neq a_i$ and $x_i \neq -1$, set $x_i = -1$.
2. If $t_i \neq -1$, go to point 5.
3. If $s_{iai} = 1$ and $g_{ai} = 0$, set $t_i = a_i$, set $t_i' = A_{ai}$, go to step 5.
4. For all j=0...J-1 execute: if $t_i \neq -1$, go to point 5; if $D_{ij} = 1$ and $s_{ij} = 0$ and $g_j = 1$, set $t_i = j$, set $t_i' = A_j$.
5. If $x_i \neq -1$, the algorithm ends.
6. According to (9) if $Avail(M_S, a, i) = 0$, set $x_i = a_i$, set $x_i' = T_{ai}$, completion of the algorithm.

The local scheduling strategy for node i populates t_i and t_i', which show the current artifact to be transferred and the amount of data that needs to be transferred to complete the transfer. The direction of transmission depends on whether this artifact is visible in the system. That is, if $g_{ti} = 1$, then the data is transmitted to the node, and if $g_{ti} = 0$, then from the node to the system.

Algorithm for the i-th node:

1. If $t_i = -1$, the algorithm ends.
2. Set $t_i' = t_i' - c_i$.
3. Set $E = E + e_i^c$.
4. If $t_i' > 0$, the algorithm ends.
5. If $g_{ti} = 1$, set $s_{i,ti} = 1$, otherwise set $g_{ti} = 1$.
6. Set $t_i = -1$, set $t_i' = 0$.

The local scheduling strategy for node i populates x_i and x_i', which show the current subtask to be computed and the number of operations required to complete the computation.

Algorithm for the i-th node:

1. If $x_i = -1$, the algorithm ends.
2. Set $x_i' = x_i' - p_i$.
3. Set $E = E + e_i^p$.
4. If $x_i' > 0$, the algorithm ends.

Set $s_{i,ti} = 1$, set $x_i = -1$, set $x_i' = 0$.

3. Experiment and research results

3.1. Experiment description

The problem of ray tracing was chosen for model verification. Ray tracing is a reference problem from the viewpoint of parallel computing. The basic idea of ray tracing is that for each pixel of the image, the algorithm models the path of a light ray, tracking its interaction with objects in the scene.

Ray tracing is ideal as a reference problem for several reasons:

– a large degree of parallelism. Each ray can be calculated independently of the others. This allows you to efficiently distribute tasks between CPU cores and thousands of GPU cores;

– a large number of calculations. Ray tracing often involves working with large volumes of data, which requires high computing power;

– the ray tracing algorithm is computationally intensive for each ray, especially when realistically rendering lighting, shadows, reflections, and refractions;

– parallel computing allows the ray tracing algorithm to scale, using more resources to speed up processing or improve image quality;

– use of specialized hardware such as GPUs (Graphic Processing Units) optimized for parallel computing, which can significantly speed up the ray tracing process;

– in summary, the use of parallel computing for ray tracing allows you to efficiently process complex scenes with a high level of realism in a relatively short time.

Another important requirement for the reference task in the context of this study is its identity on different platforms. For an adequate comparison of energy costs, it is necessary that the volume of calculations on each distributed system is completely identical. Therefore, it was decided to develop our own implementation of ray tracing for the following platforms: CPU (support for x86-64, arm64 architectures), Apple Metal and NVidia CUDA. The C++ programming language and its specific

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

variants for Metal and CUDA were used.

For simplicity, the ray tracing algorithm only works with spheres and considers diffuse lighting, directional lighting, and reflections from the material properties.

NVidia GeForce RTX 3090 graphics card (traditional server equipment) and iPhone 12 mini (mobile equipment) were used as the hardware. Standard Ethernet is used for data exchange between the desktop computer and the control center, that is, the control center and the computer will be physically close and on the same network. The mobile Internet is used for data exchange between mobile nodes, and the control center is deployed in the data center. Such conditions are typical considering the principle of operation of the corresponding distributed systems.

The simulation of multiple nodes in a distributed computer system is implemented with virtual nodes: each node is an isolated virtual entity. If one node physically has the artifact and another does not, the first one still should download it as it was a dedicated physical node. This approach is still accurate from an energy consumption perspective because all operations and calculations are conducted in a similar way on a system of physically dedicated nodes. However, the computation time depends on the amount of computing resources in the system, but accurate time estimation is not the purpose of this work.

Since the purpose of this study is to compare the energy efficiency of a distributed system based on traditional server hardware and mobile devices, it is not necessary to describe the complexity of the task in elementary operations. The main thing is that the same measure is used to assess the difficulty of the task. Given the completely identical implementation on all platforms, in this case, we choose the number of rays per unit of energy as the unit of efficiency.

The standard resolution of the output renders is the so-called "2k" and "4k" or 3840x2160 pixels. 64 rays are traced for each pixel. Thus, for "4k" it is about 530 million rays are traced for the rendering of one frame, for "2k" – 133 million rays.

Energy consumption is calculated as follows.

1. The GPU Z program is used to calculate the energy consumption of the graphics adapters. The average background energy consumption in an unloaded state is measured, followed by the consumption during the execution of a computing task. The measurements are summed, and the instantaneous power is multiplied by the time between the measurements and the obtained result. The costs of the central processor in this case can be neglected. The accumulated energy is converted into the number of rays per joule.

2. To calculate the energy consumption of a mobile device, the device is charged to 100% and the total battery discharge is calculated. The battery capacity is taken as the nominal value for the mobile device multiplied by the state of the battery. When the device is turned off, we measure the number of calculated frames and convert it into the number of rays per joule.

3. To calculate the energy consumption of the central processor of a stationary computer, we use a method similar to that of a stationary graphics processor, but with different software. To calculate the energy consumption of energy consumption on the central processor of a mobile device, an approach that is completely identical to that of a mobile graphics processor is used.

The main disadvantage of this method is that it does not consider the costs of a stationary computer for data transfer. However, they can be considered insignificant. At the same time, energy costs for data transmission will be taken into account and are expected to be significant.

3.2. Calculation of the consumption of the computing system based on the model

To calculate the energy consumption in distributed system computing, the parameters of the computing process must be entered for each system: the computing system model, the computing task model (or work model), and the computing strategy model.

A model of the computer system configuration:

– let the simulation duration be 1 s;

– the number of nodes C for this distributed problem is not of great importance, because both the problem itself and the principle of parallelism do not impose restrictions on this parameter. Since we emulate multiple virtual nodes with a single physical node, the actual number of nodes C > 1 will not affect the amount of energy we spend, but it will affect computing time estimation. To reach some accuracy in time estimation, we can assume C=1, but with the condition that no data for the calculation of the problem will be transferred between nodes and that all artifacts are always loaded into the system; however, calculations of energy consumption for C>1 will be provided;

– availability n_i node is a dynamic value with an initial value of 0;

– R_i^{on} and R_i^{off} will be selected on the basis that a mobile device with a probability of 0.5 loses connection once every 100 s:

$$(1-R_j^{on})^{100}=0.5;\ R_j^{on}=1-\sqrt[100]{0.5}=0.0069075046;$$

and returns online with a probability of 0.5 within 10 s:

$$(1-R_j^{on})^{10}=0.5;\ R_j^{on}=1-\sqrt[10]{0.5}=0.0669670085;$$

these values are selected to simulate devices that are

online most of the time and drop off for comparatively smaller periods of time;

– computing power p_i = 62,955,597 p/t (rays per current);

– bandwidth c_i = 1.2 Mb/t (megabyte per tick) Energy consumption per current during calculation e_i^p = 2.91 J;

– energy consumption per stream during data transmission e_i^c = 0.43 J;

– values p_i, c_i, e_i^p and e_i^c calculated by the benchmark execution method. For this, the mobile device (iPhone 12 mini) was fully charged, after which the device traced the scene until it was fully discharged. Nominal battery capacity 8.57 Wh = 8.57 3600 = = 30852 J. The battery life of the test device is estimated to be 81% of the nominal capacity, i.e. 24990 J. For the benchmark, a scene from a real task was chosen, and the resolution was the same: 4k and 64 rays per pixel. Thus, 530 million rays are traced to calculate one frame. The full discharge took 8572 seconds, 1017 frames were rendered. The rendering time of one frame is 8.43s, so every second the mobile device traces about 63 million rays consumes 2.91 J of energy. A comparable test was conducted for data downloading via HTTPS. The bandwidth when transmitting consecutive packets of 500 KB via mobile Internet without saving the connection is 1.2 Mb /s, the device was fully discharged in 57813 seconds, which means that the device consumed an average of 0.43 W.

A model of the computer system configuration:

– let the simulation run also be 1 s;

– the number of nodes C for a given distributed task is also not very important. Therefore, for simplicity, we consider C=1;

– availability of the n_i node is a dynamic value, but for stationary equipment it is considered that the node is always available, therefore n_i = 1;

– the hardware of a datacenter is considered reliable hence R_i^{on} = 1 and R_i^{off} = 0;

– computing power p_i = 1,271,476,886 p/t (rays per current);

– bandwidth c_i = 10 Mb/t (megabyte per tick);

– energy consumption per current during calculation e_i^p = 205 J;

– energy consumption per stream during data transmission e_i^c = 0 J;

– values p_i, c_i, e_i^p and e_i^c calculated by the benchmark execution method. For this, the GeForce RTX 3090 stationary graphics accelerator rendered 800 frames with a resolution of 3840x2160 pixels with 64 rays per pixel. Rendering was performed with power tracking enabled per second. Rendering took 334 seconds and used 68,483 joules of energy. The average frame rendering time is 0.42s. Every second, the graphics adapter traces about 760 million rays and consumes 218 joules of energy.

The distributed work in this case is general and represents the rendering of 600 frames. Each frame requires a scene to render, and the result is a frame with a resolution of 3840x2160. A typical scene is a 5 kb JSON file, a typical JPEG frame is 450 kb. Rendering occurs at a density of 64 rays per pixel. That is, to perform the task of rendering one frame, you need to download 5 KB from the system, trace 530 million (3840·2160·64) rays, and upload the 450 KB result to the system. This is represented in the system as follows: we have 600 scene generation service tasks that have no input artifacts and do not need to be executed, and 600 real frame rendering tasks, each of which depends on one artifact and each task generates one artifact:

– for $0 \le i < 600$ T_i = 0 rays;

– for $0 \le i < 600$ A_i = 5 kb;

– for $600 \le i < 1200$ T_i = 530,000,000 rays;

– for $600 \le i < 1200$ A_i = 450 kb;

– for $i = j + 600$ D_{ij} = 1, otherwise D_{ij} = 0;

– J = 1200.

As a distribution strategy, we use a naive heuristic distribution strategy.

Let's create a model of a distributed computing problem. The input (the scene) is an artifact, and the result of the rendering is also an artifact. Each task (task) contains information about computing resources in millions of rays and contains identifiers of input and output data (artifact). The program implementation is given in Listing 1.

Listing 1

Composing a program representation of Job model

```
void Job::generateRT(Job& job,
                     int frames,
                     int download,
                     int upload,
                     int oops)
{
  for (int i=0; i<frames; ++i)
  {
    Artifact a_in;
    a_in.id = job.artifacts.size();
    a_in.type =
Artifact::Type::Input;
    a_in.data = download ;
    // The amount of data
    // to download is 5 kb

    job.artifacts.push_back(a_in);

    Artifact a_out;
    a_out.id = job.artifacts.size();
    a_out.type =
```

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

```
Artifact::Type::Output ;
    a_out.data = upload;
    // Volume of output data
    // 450 kb

   job.artifacts.push_back(a_out);

    Task task;
    task.id = job.tasks.size();
    task.ops = ops;
    // computational complexity
    // 530 million rays

   task.depIds.push_back(a_in.id);
    // Reference to input data
    task.outputId = a_out.id;
    // Reference to output data
    job.tasks.push_back(task);
  }
}
```

A model of a computing node based on mobile devices is given in Listing 2.

Listing 2

Composing a program representation of a mobile device as a computing node in a distributed system

```
static NodeConfig
iPhone12miniIdealNodeConfig()
{
  NodeConfig nodeConfig;
  nodeConfig.rateOn = 1;
  // Within the framework
  // of this test
  // it is assumed that the node
  nodeConfig.rateOff = 0;
  // always reachable for
  // calculations
  nodeConfig.networking = 1200;
  // kilobytes per second
  nodeConfig.computing = 63;
  // Computational resources
  // million rays per stream
  nodeConfig.eNetworking = 430;
  // Millijoules
  nodeConfig.eComputing = 2910;
  // Millijoules
  return nodeConfig;
}
```

Computational simulation results based on one mobile device for 1017 frames: 10171 ticks and 25062 Joules of expenditure. The execution time differs from the benchmark because of the peculiarities of the system implementation: any task is performed for an integer number of ticks, therefore, instead of 8.43 seconds, the model spends 9 ticks on the task. If the task is completed on a given log, the new one will be executed only on the next one. In addition, the naive task allocation strategy allocates a task to a node only when the node is free. Immediately after the distribution, the node starts downloading the input data, which lasts at least 1 thread. Thus, we have 1 additional thread for each task + 1 thread for downloading the last artifact to the system. If we allow the naive strategy to add a task to the node's queue, each task will be executed in 9 ticks because the node will download data for the tasks in the queue. However, it does not affect energy consumption, so complicating the strategy within the scope of this work is not advisable. The energy consumption in the simulation is also different, but only because the model considers data transmission over the network. If we set the energy costs for energy transfer to 0, we obtain a value of 24896 joules, which is 0.4% less than the measured value. The number of computing nodes in the simulation does not affect the energy consumption, which makes sense.

The calculation model of the processing node based on stationary computers is given in Listing 3.

Listing 3

Composing a program representation of a stationary GPU as a computing node in a distributed system

```
static NodeConfig
rtx3090NodeConfig()
{
  NodeConfig nodeConfig ;
  nodeConfig.rateOn = 1;
  nodeConfig.rateOff = 0;
  nodeConfig.networking = 10000;
  // Bandwidth is considered
  // high enough
  nodeConfig.computing = 1271;
  // Computing resources
  nodeConfig.eNetworking = 0;
  // Data transfer costs are
considered
  // insignificant
  nodeConfig.eComputing = 205000;
  // Millijoules
  return nodeConfig;
}
```

Because of simulating (Table 2) a distributed calculation of 800 frames, we received 1601 ticks and 68386 joules of energy. The length in ticks is also calculated by rounding up + 1 tick per download + 1 last tick. The obtained energy consumption deviates from the measured value by 0.2%.

Table 2

Comparison of characteristics of mobile and stationary GPUs for 4k and 2k frames

	Measurement	Forecast	Deviation
Execution time 4k (stationary)	276 s	1200 s	335%
Execution time 4k (mobile)	6061 s	6696 s	10.5%
Consumption 4k (stationary)	52806 J	51.3 kJ, C=1	2.7%
		51.3 kJ, C=10	
		51.3 kJ, C=50	
Consumption 4k (mobile)	15320 J	15.1 kJ, C=1	1.6%
		15.1 kJ, C=10	
		15.1 kJ, C=50	
Execution time 2k (stationary)	71 s	1200 s	-
Execution time 2k (mobile)	6122 s	7007 s	14%
Consumption 2k (stationary)	53337 J	54.5 kJ, C=1	2.2%
		54.5 kJ, C=10	
		54.5 kJ, C=50	
Consumption 2k (mobile)	15320 J	15.2 kJ, C=1	1%
		15.2 kJ, C=10	
		15.2 kJ, C=50	

The error in the calculated energy consumption in both cases comes from the fact that both the computational complexity of the tasks and the computing power of the devices are represented by whole numbers for convenience and speed.

The physical computation of 600 frames on a distributed system will occur on a single device that will simulate different compute nodes so that it will not maintain connections and will not use input data from other computations to possibly avoid overhead. Failure simulation is performed using the node software. In addition, considering the peculiarity of the battery, the calculation will proceed from a charge level of 80%. Running the calculation on the system from a single mobile device reduced the battery charge to 19%, thus consuming 15,320 joules of energy. The calculation took 6061 s. The computation of 600 frames on a distributed system with stationary equipment took 276 s and consumed 52806 joules of energy.

A simulation of computing 600 frames on a mobile distributed system with failover settings shows 6696 ticks and 15078 joules of energy. A simulation of computing on a distributed system from stationary devices shows 1200 ticks and a consumption of 51290 joules of energy. Changing the number of nodes in the simulation does not change the energy consumption, which is expected. Only the general computation time is changed in this case.

The result of a typical scene rendered with ray tracing is shown in Figure 1.

4. Discussion

The significant variance in execution time on stationary devices can also be attributed to the model's discrete nature and specific characteristics. To obtain a more accurate forecast, it is necessary to obtain a lower resolution of the simulation. However, estimating the execution time is not the purpose of the work.

The smaller deviation of the execution time of the mobile model compared to the benchmark is due to the fact that each frame on the mobile device is now processed for almost exactly 9 seconds, which reduces the error of the discrete model.

A distributed system based on mobile devices consumes 3.45 times less energy for the same calculations. Compared to the difference of 2.2 times in the previous work [20], this is a lot, but there are several explanations for this. Initially, the prior study contrasted the same mobile device against the GeForce RTX 3080, noted for its higher energy efficiency compared to the 3090. Second, the previous work used a synthetic benchmark that loaded only the graphics card processor, without loading the memory at all. In this case, the memory is used more intensively, and the RTX 3090 memory controller consumes more than the RTX 3080. Third, the benchmark in this work is worse optimized for CUDA than for Metal, but it would be more accurate to say that Metal is more "forgiving" to the developer. Fourth, the graphics card of a desktop computer has additional consumers, such as an active cooling system, and their consumption has been considered.

A better implementation of ray tracing can reduce the gap between stationary and mobile equipment, but you can't expect equal or close power consumption to do the same job.

Conclusions

This paper presents the development and testing of a universal model for predicting energy costs during calculations that can be applied to both a traditional

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

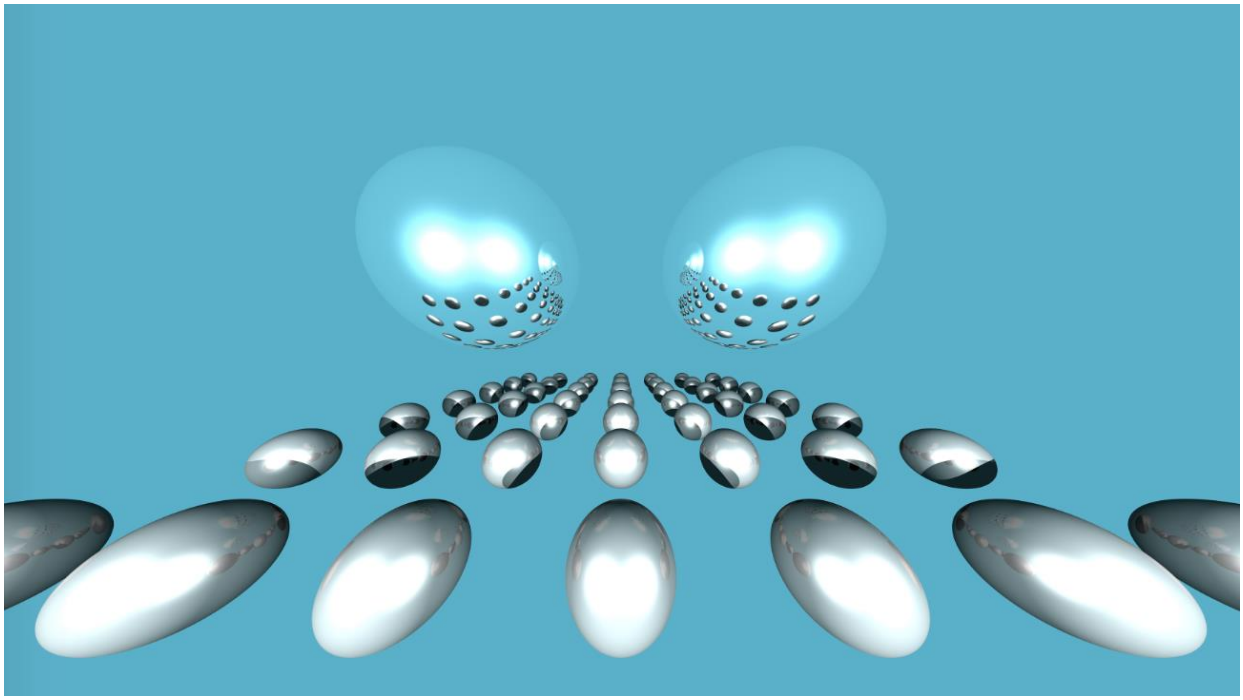

Fig 1. Rendering result

stationary computer system and a system based on mobile devices.

The approach proposed in this study allows us to evaluate energy efficiency without specialized measuring equipment, determine the energy efficiency of GPUs based on CUDA technology, evaluate the impact of various parameters on performance and energy efficiency, and compare them with a similar implementation using Apple Metal GPUs. The results of the experiments conducted by the author show that the energy consumption assessment model considers the features of the hardware platform on which data processing is carried out. Thus, the GPU implementation using CUDA technology with small data packet sizes, such as 1 input vector and even 1000 input vectors, is inferior in energy efficiency to a similar implementation on the Apple Metal GPU.

Therefore, during experimental testing, the energy consumption estimation model turned out to be quite effective. However, the model requires a clear goal for both the task and the device used. This, in turn, requires the creation of a benchmark for each model, which is a disadvantage of this model. This drawback is planned to be eliminated by accumulating quantitative data and modeling results, which will allow generalizing some parts of the model. In addition, the task in this work was chosen quite ideally: it does not require intensive data exchange between computing nodes.

Including future research directions, it is planned to apply the model to more coherent calculations. Thus, the presented results and the described measurement methods can be used as a basis for conducting large scale studies in the field of assessing the energy consumption of mobile and stationary GPUs.

Another direction for future research is information security. Calculations on personal mobile devices require additional information security solutions because it is impossible to physically protect the hardware from unauthorized access.

Contributions of authors: conceptualization, methodology, analysis development of model, software, verification visualization, writing original text – **Oleksandr Mamchych**; scientific supervising, analysis of results, reviewing and editing – **Maksym Volk.**

Conflict of interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

This research was conducted without financial support.

Data availability

Manuscript has no associated data.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence methods while creating the presented work.

All the authors have read and agreed to the published version of this manuscript.

References

1. Javed, A., Alyas Shahid, M., Sharif, M., & Yasmin, M. Energy consumption in mobile phones. *International Journal of Computer Network and Information Security*, 2017, vol. 9, no. 12, pp. 18-28. DOI: 10.5815/ijcnis.2017.12.03

2. Roth, K., Goldstein, F., & Kleinman, J. *Energy consumption by office and telecommunications equipment in commercial buildings volume I: energy consumption baseline*, National Technical Information Service (NTIS), US Department of Commerce, Springfield, 2002. 211 p. Available at: https://biblioite.ethz.ch/downloads/Roth_ADL_1.pdf (Accessed 01.03.2024)

3. Giri, A., & Patil, P. Design of a parallel multi-threaded programming model for multicore architecture with resource sharing. *Indian Journal of Scientific Research*, 2015, vol. 11, no. 1, pp. 85-89. Available at: https://go.gale.com/ps/i.do?id=GALE%7CA454619960&sid=googleScholar&v=2.1&it=r&linkaccess=abs&issn=09762876&p=AONE&sw=w&userGroupName=anon%7Eaed63acc&aty=open-web-entry (Accessed 01.03.2024)

4. Caspart, R., Ziegler, S.; Weyrauch, A.; Obermaier, H., Raffeiner, S., Schuhmacher, Leon P., Scholtyssek, J., Trofimova, Darya., Nolden, M., Reinartz, I., Isensee, F., Götz, M., & Debus, C. Precise energy consumption measurements of heterogeneous artificial Intelligence workloads. *ISC High Performance 2022: High Performance Computing. ISC High Performance 2022 International Workshops*, 2022, pp. 108-121. DOI: 10.1007/978-3-031-23220-6_8.

5. Chandrakasan, A. P., Sheng, S., & Brodersen, R. W. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 1992, vol. 27, no. 4. pp. 473–484. DOI: 10.1109/4.126534.

6. Dong, M., & Zhong, L. Self-constructive high-Rate system energy modeling for battery-powered mobile systems. *ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys'2011)*, Association for Computing Machinery, New York, NY, USA, pp. 335–348. DOI: 10.1145/1999995.2000027.

7. Saipullah, K. M., Anuar, A., Atiqah Ismail, N., & Soo, Y. Measuring power consumption for image processing on android smartphone. *American Journal of Applied Sciences*. 2012, vol. 9, no. 12, pp. 2052–2057. DOI: 10.3844/ajassp.2012.2052.2057.

8. Bekaroo, G., & Santokhee, A. Power consumption of the Raspberry Pi: A comparative analysis. *In 2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*. Balaclava, Mauritius, 2016, pp. 361-366. DOI: 10.1109/EmergiTech.2016.7737367.

9. Dean, J., & Ghemawat, S. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 2008, vol. 51, no. 1, pp. 107-113. DOI: 10.1145/1327452.1327492.

10. Carvalho, S. A., Lima, R. N., Cunha, D. C., & Silva-Filho, A. G. A hardware and software web-based environment for Energy Consumption analysis in mobile devices. *In 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, 2016, pp. 242–245. DOI: 10.1109/NCA.2016.7778625.

11. Kharchenko, V., Brezhnev, E., & Sklyar, V. Green information technologies: paradigm and cooperation in research, development and education domains. *8th International Green Energy Conference*, Kyiv, Ukraine, 2013, pp. 1-5. https://www.researchgate.net/publication/305787567_Green_Information_Technologies_Paradigm_and_Cooperation_in_Research_Development_and_Education_Domains (Accessed 01.03.2024)

12. Kharchenko, V., Gorbenko, A., Sklyar, V., & Phillips, C. Green computing in critical application domains: challenges and solutions. *10th Conference on Digital Technologies, DT2013*. Žilina, Slovakia, 2013, pp 191-197. DOI: 10.1109/DT.2013.6566310

13. Mamchych, O., & Volk, M. Smartphone based computing cloud and energy efficiency published in: 2022. *12th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Athens, Greece, 2022. DOI: 10.1109/DESSERT58054.2022.10018740.

14. Hamza, S. Distributed computing system on a smartphones-based network *in book: Software Technology: Methods and Tools*, 2019, pp. 313-325. DOI: 10.1007/978-3-030-29852-4_26.

15. Yu, J., Williams, E., & Ju, M. Analysis of material and energy consumption of mobile phones in China. *Energy Policy*, 2010, vol. 38, no. 8, pp. 4135-4141. DOI: 10.1016/j.enpol.2010.03.041.

16. Damaševičius, R., Štuikys, V., & Toldinas, J. Methods for measurement of energy consumption in mobile devices. *Metrology and measurement systems*, 2013, vol. 20, no. 3, pp. 419-430. DOI: 10.2478/mms-2013-0036

17. Comito, C., & Talia, D. Energy consumption of data mining algorithms on mobile phones: Evaluation

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

and prediction. *Pervasive and Mobile Computing*, 2017, vol. 42, pp. 248-264. DOI: 10.1016/j.pmcj.2017.10.006

18. Fekete, K., Csorba, K., Forstner, B., Fehér, M., & Vajk, T. Energy-efficient computation offloading model for mobile phone environment. *In 2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, Paris, France, 2012, pp. 95-99. DOI: 10.1109/CloudNet.2012.6483662.

19. Ramos, R., Faria P., Gomes L., & Vale Z. Building Energy Consumption Forecast under Different Anticipations on a Green Computation Perspective *IFAC-PapersOnLine*, 2023, vol. 56, Issue 2, pp. 10923-10928. DOI: 10.1016/j.ifacol.2023.10.778.

20. Lee, K. P., Chng, C.W, Tong, D.L., & Tseu, K. L. Optimizing Energy Consumption on Smart Home Task Scheduling using Particle Swarm Optimization. *Procedia Computer Science*, 2023, vol. 220, pp. 195-201. DOI: 10.1016/j.procs.2023.03.027.

21. Catthoor, F., Wuytack, S., De Greef, E., Balasa, F., Nachtergaele, L., & Vandecappelle, A. *Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design*. Boston, Kluwer Academic Publishers, 1998. 356 p.

22. Ivanisenko, I. M., & Volk, M. O. Simulation methods for load balancing in distributed computing. *Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2017)*, Novi Sad, Serbia, 2017, pp. 690-695. DOI: 10.1109/EWDTS.2017.8110078.

23. Kondratenko, Y., Kozlov, O., Korobko, O., & Topalov, A. Complex industrial systems automation based on the internet of things implementation. *Communications in Computer and Information Science*, Springer, Cham, 2018, pp. 164–187. DOI: 10.1007/978-3-319-76168-8_8.

24. Kondratenko, Y. P., Kozlov, O. V., Korobko, O. V., & Topalov, A. M. Internet of Things approach for automation of the complex industrial systems. *13th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer*, Kyiv, Ukraine, 2017, pp. 3-18. Available at: https://ceur-ws.org/Vol-1844/10000003.pdf (Accessed 01.03.2024)

25. Güçyetmez, M., & Farhan, H.S., Enhancing smart grids with a new IOT and cloud-based smart meter to predict the energy consumption with time series. *Alexandria Engineering Journal*, 2023, vol. 79, pp. 44-55. DOI: 10.1016/j.aej.2023.07.071.

26. Zakaria, S., Mativenga P., & Ariff, E.A.R Engku. An Investigation of Energy Consumption in Fused Deposition Modelling using ESP32 IoT Monitoring System. *Procedia CIRP*, 2023, vol. 116, pp. 263-268. DOI: 10.1016/j.procir.2023.02.045.

27. Cheng-Fu, H., Ding-Hsiang, H., & Yi-Kuei, L. Network Reliability Evaluation for a Distributed Network with Edge Computing. *Computers & Industrial Engineering*, vol. 147, 2020. DOI: 10.1016/j.cie.2020.106492.

28. Guobin, Z., Jian, Z., Jian, T., & Junwu, Z. Collaboration Energy Efficiency with Mobile Edge Computing for Data Collection in IoT. *Advances in Artificial Intelligence and Security*, Beijing, 2021, pp. 279-285. DOI: 10.1007/978-3-030-78615-1_24.

29. Qasaimeh, M., Denolf, K., Lo, J., & Vissers, K. Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels, *The 15th IEEE International Conference on Embedded Software and Systems*, Nevada, US, 2019, pp. 4-8. DOI: 10.1109/ICESS.2019.8782524.

Received 27.02.2024, Accepted 15.04.2024

УНІФІКОВАНА МОДЕЛЬ ТА МЕТОД ПРОГНОЗУВАННЯ ЕНЕРГОСПОЖИВАННЯ В РОЗПОДІЛЕНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ НА ОСНОВІ СТАЦІОНАРНИХ ТА МОБІЛЬНИХ ПРИСТРОЇВ

О. О. Мамчич, М. О. Волк

Предметом дослідження в даній статті є прогнозування енергоспоживання при обчисленні розподілених завдань у комп'ютерних мережах, побудованих на базі серверних рішень та розподілених персональних смартфонів. **Метою** цього дослідження було створення універсальної моделі прогнозування вартості обчислювальної енергії, яку можна застосувати як до традиційної хмарної системи, так і до мобільної хмарної системи. **Завдання**: провести аналіз енергозберігаючих підходів і технологій, що використовуються для розрахунку даних; розглянути моделі комп'ютерної системи та дії з ними, а саме: модель розподіленої роботи, модель стратегії розподілу, ініціалізацію моделі; розробити єдиний і єдиний динамічний метод прогнозування витраченої енергії з акцентом на гетерогенні системи; проведення дослідження запропонованого підходу на стаціонарних та мобільних пристроях. Отримані **результати** роботи включають результати експериментальних вимірювань енергоспоживання мобільних цифрових систем та стаціонарних. Була визначена енергоефективність обчислень на графічних процесорах стаціонарного пристрою на основі

технології CUDA та графічних процесорів на мобільних пристроях на основі технології Apple Metal. Обчислення 600 кадрів в розподіленій системі з мобільних пристроїв з налаштуваннями відмов показало споживання 15078 джоулів енергії. Моделювання обчислень на розподіленій системі зі стаціонарними пристроями показало споживання 51290 джоулів енергії. Це означає що мобільна система дає виграш у 3,45 разів в споживанні енергії. **Висновки.** Модель оцінки енергоспоживання виявилася досить ефективною. Результати авторських експериментів показують, що модель оцінки енергоспоживання враховує особливості апаратної платформи, на якій виконується обробка даних. Обчислення даних на GPU стаціонарних пристроїв програє в енергоефективності аналогічній реалізації на GPU Apple Metal з мобільних пристроїв. Отже, представлені результати доводять раціональність використання мобільних графічних процесорів для енергоефективної обробки інформації.

Ключові слова: графічний процесор; енергоефективність; розподілена система; хмарні обчислення; зелені обчислення; мобільний пристрій; багатопоточність.

Мамчич Олександр Олександрович – асп. каф. Електронних Обчислювальних Машин, Харківський Національний Університет Радіоелектроніки, Харків, Україна.

Волк Максим Олександрович – д-р техн. наук, проф. каф. Електронних Обчислювальних Машин, Харківський Національний Університет Радіоелектроніки, Харків, Україна.

Oleksandr Mamchych – PhD Student at the Department of Electronic Computers (ECM), Kharkiv National University of Radioelectronics, Kharkiv, Ukraine,
e-mail: mamont0207@gmail.com, ORCID: 0009-0001-6602-2929, Scopus Author ID: 58099399400.

Maksym Volk – Doctor of Technical Science, Professor at the Department of Electronic Computers (ECM), Kharkiv National University of Radioelectronics, Kharkiv, Ukraine,
e-mail: maksym.volk@nure.ua, ORCID: 0000-0003-4229-9904, Scopus Author ID: 9636701100.