Article

The Software Cache Optimization-Based Method for Decreasing Energy Consumption of Computational Clusters

Alla G. Kravets [1,2,*] and Vitaly Egunov [3]

[1] CAD&RD Department, Volgograd State Technical University, 400005 Volgograd, Russia
[2] Institute of System Analysis and Management, Dubna State University, Moscow Region, 141982 Dubna, Russia
[3] Computers and Systems Department, Volgograd State Technical University, 400005 Volgograd, Russia
* Correspondence: agk@gde.ru

Abstract: Reducing the consumption of electricity by computing devices is currently an urgent task. Moreover, if earlier this problem belonged to the competence of hardware developers and the design of more cost-effective equipment, then more recently there has been an increased interest in this issue on the part of software developers. The issues of these studies are extensive. From energy efficiency issues of various programming languages to the development of energy-saving software for smartphones and other gadgets. However, to the best of our knowledge, no study has reported an analysis of the impact of cache optimizations on computing devices' power consumption. Hence, this paper aims to provide an analysis of such impact on the software energy efficiency using the original software design procedure and computational experiments. The proposed Software Cache Optimization (SCO)-based Methodology was applied to one of the key linear algebra transformations. Experiments were carried out to determine software energy efficiency. RAPL (Running Average Power Limit) was used—an interface developed by Intel, which provides built-in counters of Central Processing Unit (CPU) energy consumption. Measurements have shown that optimized software versions reduce power consumption up to 4 times in relation to the basic transformation scheme. Experimental results confirm the effectiveness of the SCO-based Methodology used to reduce energy consumption and the applicability of this technique for software optimization.

Keywords: energy efficiency of software; RAPL; cache memory; software cache optimization; reflection transformation; householder transformation; cache miss; analytical efficiency evaluation

Citation: Kravets, A.G.; Egunov, V. The Software Cache Optimization-Based Method for Decreasing Energy Consumption of Computational Clusters. *Energies* **2022**, *15*, 7509. https://doi.org/10.3390/en15207509

Academic Editor: Abu-Siada Ahmed

Received: 15 September 2022
Accepted: 9 October 2022
Published: 12 October 2022

1. Introduction

In the modern world, the issue of energy efficiency of various classes and architectures of computing systems is extremely important. This applies equally to mobile phones [1], which are quite powerful computing systems, and various embedded devices [2], as well as laptops, computers, and entire computing clusters [3]. In fact, it is extremely important for the customer to use a mobile phone, laptop, or other smart gadgets for a long time without recharging [4]. The issue of computers and computing clusters' energy efficiency is mainly related to the electricity consumed and bills for it. Additionally, if the individual computers' energy efficiency is not so relevant, then the high energy consumption of computing clusters can lead to huge electricity bills [5,6], an increase in the cost of renting computing resources, etc. It is also worth mentioning that, with an increase in electricity consumption, the device heating increases, and more effort must be spent on cooling it. In some cases, high power consumption makes the use of a particular device or software ineffective, and sometimes completely useless. As an example, we can give the mining of cryptocurrencies at a high electricity cost [7].

Accordingly, reducing the electricity consumption by computing devices is an urgent task. Traditionally, the solution to this problem has been attributed to the design of more cost-effective equipment, i.e., chip manufacturing technology improvement. However, recently there has been an increased interest in this issue on the part of software developers [8]. The issue of developing energy-efficient software can be considered from at least two points of view:

- energy-efficient software development for mobile phones [9,10] and other smart gadgets [11,12]; the goal here is to extend the operation of the device without recharging the battery [13];
- energy-efficient software development for the implementation of computationally complex algorithms for solving scientific problems, and performing engineering calculations; the goal here is to reduce energy consumption in general during solving a specific task or group of tasks [14,15].

Currently, a wide range of researchers are studying the link between the developed software and the electricity consumed by the computing system. As an example, we can cite the works already mentioned in this article, in addition, it is necessary to note the works [16,17], and some others. In these works, the issues of energy efficiency of data structures used, programming methods for various computing architectures [18], and issues related to the prediction of various software systems' energy efficiency are investigated [19,20].

In [8,21], the issue of various programming languages' energy efficiency is investigated. The authors try to answer the question of whether the fastest program is the most energy efficient [22,23]. Several research papers give different answers to this question [24,25].

There are at least two groups of research papers analyzing the energy consumption of computing systems and ways to reduce it:

- related to the energy consumption measurement, the evaluation of software energy efficiency [26];
- aimed at improving software energy efficiency [22,23].

However, to the best of our knowledge, no study reported an analysis of the impact of cache optimizations on computing devices' power consumption. Hence, this paper aims to provide an analysis of such impact on the software energy efficiency using the original software design procedure and computational experiments.

This work is organized as follows: Section 2 contains an overview of the basic transformation for optimization, Section 3—proposed Software Cache Optimization-based methodology (SCO-based methodology), and Section 4—results and discussions.

2. Basic Transformation

We have chosen one of the typical matrix transformations as the basic transformation. On the one hand, many linear algebra problems are widely used in solving many applied and research problems. Such tasks include solving systems of linear algebraic equations, searching for eigenvalues and eigenvectors systems, performing singular value decomposition, and many others. They often arise during solving, as higher-level task stages. Moreover, typical matrix transformations used in solving basic linear algebra problems have a fairly high computational complexity, especially when it comes to large-dimensional matrices, which is typical for modern high-precision and large-scale mathematical models.

Separately, we want to note some two-sided transformations that are widely used in linear algebra. Expression (1) shows similarity transformation.

$$B = F^{-1}AF \tag{1}$$

Similar matrices have some characteristics that coincide, in particular, rank, determinant, eigenvalues, and some others. Expression (2) shows the unitary two-sided transformation.

$$B = UAT \tag{2}$$

Here U and T are unitary matrices. This conversion also preserves some original matrix characteristics, in particular the singular values. Due to the fact that transformations (1) and (2) preserve eigenvalues and singular values, respectively, they are often used in algorithms for their calculation. The tasks themselves are well known [27,28].

The general form matrix is often reduced to an "almost triangular" form in solving the eigenvalue problem. This form differs from the triangular one by the presence of a nonzero diagonal under or above the main one. Matrices of this type are called, respectively, the upper or lower Hessenberg matrix. There is a finite transformation sequence that leads the general form matrix to a Hessenberg form [29] while preserving its eigenvalues. Such matrices, along with tridiagonal ones, are widely used in algorithms for calculating eigenvalues that strengthen the main diagonal. For example, in the QR algorithm or the Jacobi method, reducing the time for matrices transformations to such forms also reduces the time for solving the eigenvalue problem as a whole. Various mathematical methods can be implemented to convert the general form matrix to a Hessenberg form. In this paper, the Householder reflection transformation will be used [30]. Despite the fact that eigenvalues and singular value decompositions and their numerical implementation have been known for a long time and have been well studied, the researchers' attention is still attracted by the special type matrices transformation [31], as well as the use of singular values decomposition in solving new technical problems, for example [32]. When studying the ways of performing these decompositions, the authors, in particular, pay attention to Hessenberg matrices [33]. Additionally, modern research is related to the Householder reflection application and implementation [34]. For the most part, publications address various applications of this algorithm in solving various scientific and applied problems, often in relation to specific computing architectures [35,36]. Among the works devoted to linear operators' optimization for modern architectures, it is possible to note works devoted to cache optimization [37,38].

Separately, it is worth noting the block Householder transformation (BH), which also receives a lot of attention from researchers.

Let Q(3) be the product of the Householder $n \times n$ matrices.

$$Q = Q_1 .....Q_R \tag{3}$$

Matrices Q can be represented as (4), where W and Y are of dimension n * r [39].

$$Q = I + WY^T \tag{4}$$

The pseudo-code of matrix Q block representation can be represented as follows.

```
1:  $Y = v^{(1)}$
2:  $W = -2\ v^{(1)}/[v^{(1)}]^T\ v^{(1)}$
3:  for $j = 2$ to $r$
4:      $z = -2(I + WY^T)\ v^{(j)}/[v^{(j)}]^T\ v^{(j)}$
5:      $W = [W\ z]$
6:      $Y = [Y\ v^{(j)}]$
7:  end
```

Here $v^{(j)}$—j-th reflection vector, and Y—matrix of reflection vectors.

The BH is based on the reflection vectors' accumulation and mainly uses Basic Linear Algebra Subprograms (BLAS) Level 3 operations (matrix–matrix operations, including matrix multiplication *gemm*). At the same time, the traditional Householder reflection is based on BLAS Level 2 operations (matrix–vector operations). At the same time, BLAS 2 operations are implemented less efficiently on modern parallel computing architectures than BLAS 3 operations. This is due to the fact that BLAS 3 operations are potentially better optimized in terms of using the cache memory.

This BH is the basis of a large number of efficient matrix transformation algorithms, including the general form matrices' reduction to Hessenberg form. In particular, the work [40] considers the issues of Householder transformation effective implementation using the WY representation, including an application for solving eigenvalues and the eigenvectors system problems; the issues of efficient memory use when performing these transformations are analyzed.

In addition to the actual research in the field of effective BH implementation, some researchers are engaged in improving the efficiency of LAPACK (Linear Algebra PACKage) algorithms [41] on various parallel computing architectures.

In modern computing systems, parallelism can be implemented at different levels, and different types of parallelism can be used, often several within the same computing architecture [42,43]. Accordingly, research is being conducted on the adaptation of well-known numerical methods for effective implementation on modern parallel architectures, or to create completely alternative algorithms, the effectiveness of implementing algorithms on various parallel architectures is evaluated, for example, in [44,45].

The target computing architectures in this paper are parallel computing systems with shared memory, so, the following can be identified as the main directions for improving software efficiency (Figure 1).

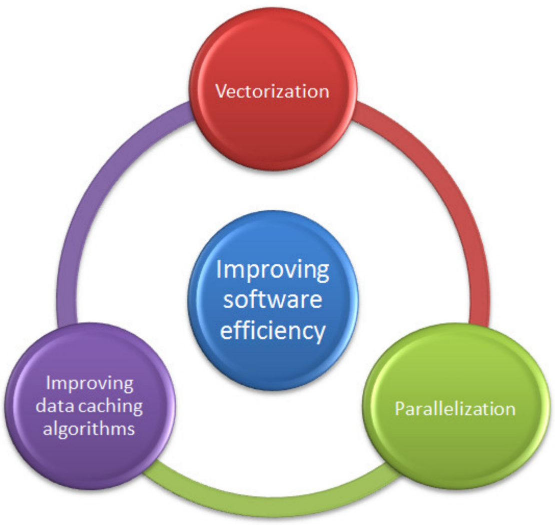
Vectorization
Improving software efficiency
Improving data caching algorithms
Parallelization

Figure 1. The main directions of increasing the software efficiency for target computing architectures.

This work is based on previous research. In particular, the paper [46] describes a vectorization method for algorithms of general form matrices eigenvalues and singular values decompositions; the Householder reflection was used as the basic transformation. Computational experiments have confirmed the effectiveness of the proposed vectorization methods. In [47], methods were proposed to improve the parallelization algorithm of the basic transformation, in particular, the method of reducing the sequential calculations proportion. This made it possible to reduce the software execution time, which proves the effectiveness of such techniques. In [48], the matrices decomposition using Householder reflections on parallel computing systems with shared memory is considered. It is shown that, in the case of performing a Householder linear transformation, it is possible to choose an optimal calculation scheme that significantly reduces the probability of cache misses. In the case of performing a two-sided conversion, the situation is not so unambiguous, since it is necessary to perform multiplication on both sides to preserve eigenvalues or singular values.

3. Software Cache Optimization-Based Methodology

This paper will show the use of SCO-based methodology, designed to improve data caching algorithms in software development for computing systems with a hierarchical memory structure (Figure 2).

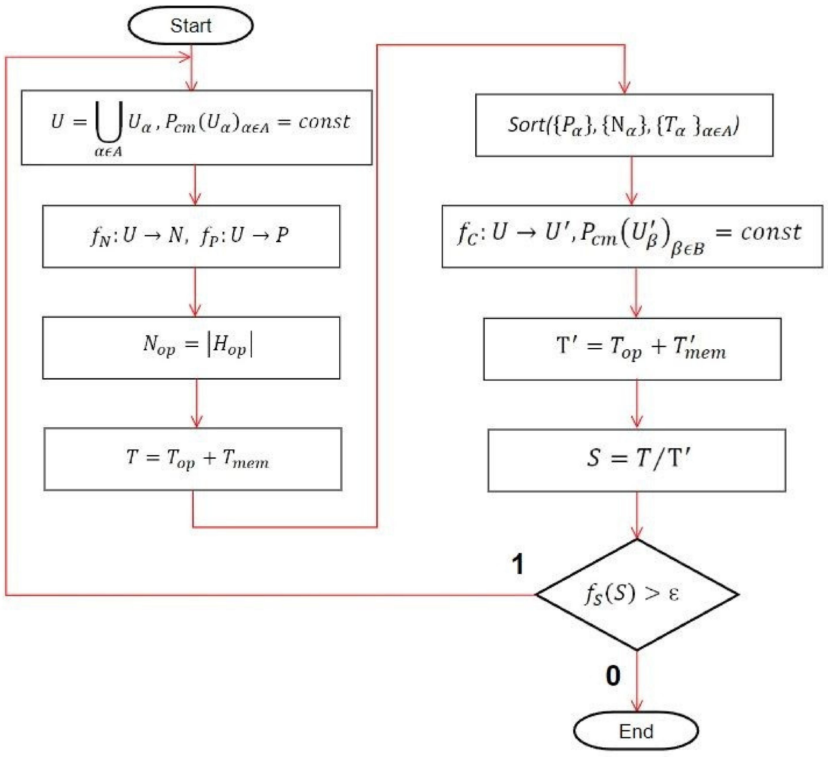
Start
$U = \bigcup_{\alpha\epsilon A} U_\alpha, P_{cm}(U_\alpha)_{\alpha\epsilon A} = const$
$f_N: U \to N,\ f_P: U \to P$
$N_{op} = |H_{op}|$
$T = T_{op} + T_{mem}$
$Sort(\{P_\alpha\}, \{N_\alpha\}, \{T_\alpha\}_{\alpha\epsilon A})$
$f_C: U \to U', P_{cm}(U'_\beta)_{\beta\epsilon B} = const$
$T' = T_{op} + T'_{mem}$
$S = T/T'$
$f_S(S) > \varepsilon$
1
0
End

Figure 2. SCO stages.

At the first SCO stage, the current required algorithm is analyzed. This algorithm is characterized by sets of arithmetic–logical operations H_{op} and memory access operations U. During analysis:

- the set of memory access operations is divided into disjoint subsets according to the method of accessing memory that affects the cache miss probability; subsets of operations with equal cache miss probability values P_{cm} are obtained;
- the number of element $\{N_\alpha\}_{\alpha\epsilon A}$ s and the cache miss probability $\{P_\alpha\}_{\alpha\epsilon A}$ are determined for each of the obtained subsets;
- the number of arithmetic and logical operations in general for the entire algorithm N_{op} is determined;
- the expression is formed that evaluates the software running time T.

At the second SCO stage, the results are analyzed and an attempt is made to correct the original algorithm to improve the caching algorithms:

- analyzing the obtained subsets of memory access operations and the corresponding values of the cache miss probability $\{P_\alpha\}_{\alpha\epsilon A}$, the number of operations $\{N_\alpha\}_{\alpha\epsilon A}$ and the execution time $\{T_\alpha\}_{\alpha\epsilon A}$ of all subset operations, the search for algorithm sections that negatively affect the data caching process efficiency is carried out;
- an attempt is made to correct the original algorithm, the result is a new division of the memory access operations set into subsets with other characteristics $\{U'_\beta\}_{\beta\epsilon B}$; permutation optimization and other approaches can be used to correct the algorithm;
- software running time T' with the modified algorithm is estimated;
- resulting speedup value S is calculated.

After completing the SCO, a decision is made to re-apply this procedure depending on the value of a quality function. For example, when making a decision, the resulting speedup value is used (Figure 2).

We applied SCO to the two-sided Householder transformation in reducing the general form matrix to the Hessenberg form. As a result, we obtained two optimized schemes for performing the calculations.

3.1. Row-Oriented Householder Reflection

Figure 3 shows the process of obtaining a row-oriented scheme for performing the basic Householder reflection algorithm. The arrows here show the sequence of row operations (solid arrows) and column operations (dotted arrows) of the original matrix within one conversion step for the reduction to the upper Hessenberg form.

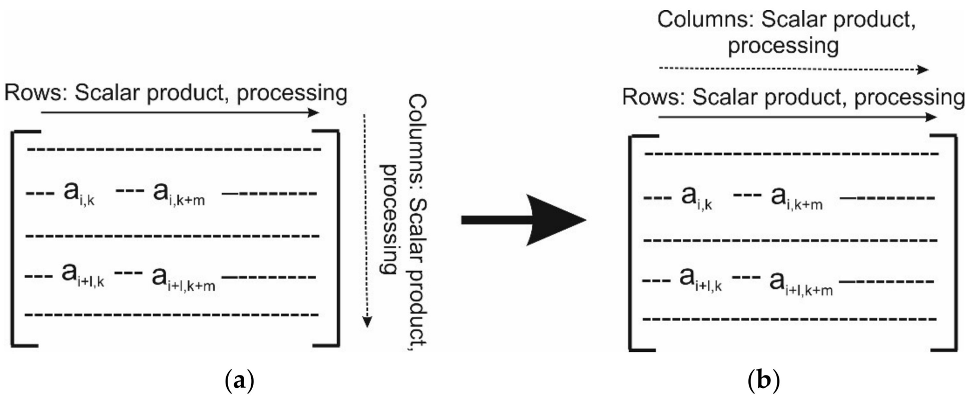
Rows: Scalar product, processing
Columns: Scalar product, processing
--- $a_{i,k}$ --- $a_{i,k+m}$ ---
--- $a_{i+l,k}$ --- $a_{i+l,k+m}$ ---
(a)
Columns: Scalar product, processing
Rows: Scalar product, processing
--- $a_{i,k}$ --- $a_{i,k+m}$ ---
--- $a_{i+l,k}$ --- $a_{i+l,k+m}$ ---
(b)

Figure 3. Obtaining a row-oriented scheme for performing a basic transformation: (**a**) traditional algorithm based on the Householder reflection; (**b**) row-oriented transformation scheme.

Figure 3a shows the scheme of the traditional algorithm based on the Householder reflection. It is extremely inefficient due to the need to process matrix columns, which generates a large number of cache misses for matrices stored along the rows.

The order of processing matrix elements could be modified. Figure 3b shows a row-oriented transformation scheme, where column processing is also carried out along the rows.

3.2. Single-Pass Householder Reflection

This algorithm contains the potential for further performance improvements. Repeated application of the same procedure shows the inefficiency of the need for two passes through the matrix to process rows and columns. As a result, at a certain dimension, by the second pass, matrix elements begin to be displaced from the cache. The matrix elements' processing order can be changed again. Now processing is carried out within one pass in the sequence first column—first row—second column—second row, etc. This approach significantly reduces the probability of cache misses. Figure 4 shows obtaining a single-pass scheme for performing the basic transformation.

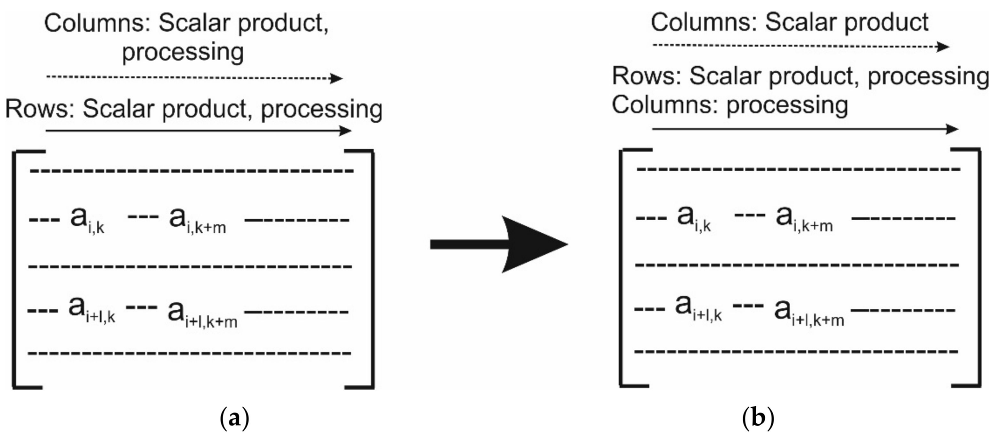
Columns: Scalar product, processing
Rows: Scalar product, processing
--- $a_{i,k}$ --- $a_{i,k+m}$ ---
--- $a_{i+l,k}$ --- $a_{i+l,k+m}$ ---
(a)
Columns: Scalar product
Rows: Scalar product, processing
Columns: processing
--- $a_{i,k}$ --- $a_{i,k+m}$ ---
--- $a_{i+l,k}$ --- $a_{i+l,k+m}$ ---
(b)

Figure 4. Obtaining a single-pass scheme for performing a basic transformation: (**a**) row-oriented transformation scheme; (**b**) single-pass transformation scheme.

The pseudo-code of this algorithm is presented below. The Hessenberg procedure is used to bring the general matrix to the upper Hessenberg form; we used the following notation: $\boldsymbol{A}$—the original matrix, N—dimension of the original matrix, Householder—the procedure for calculating the reflection vector, $\boldsymbol{u_S}$—reflection vector, g—normalization coefficient. *SPH*—procedure for performing one step of a single-pass Householder transformation (Single-Pass Householder). Matrices and vectors are highlighted in bold, and the ranges of index changes are indicated in parentheses.

```
1:   Hessenberg ($\boldsymbol{A}$, $N$)
2:       for $s = 1$ to $N - 2$ do
3:           ($g$, $\boldsymbol{u_S}$) = Householder($\boldsymbol{A_s}$, $s$);
4:           SPH($A$, $N$, $s$, $g$, $\boldsymbol{u_S}$)
5:       end for
6:   end Hessenberg
```

```
1:   SPH($A$, $N$, $s$, $g$, $\boldsymbol{u_S}$)
2:       for $r = 1$ to $(s-1)$ do
3:           $vr_s = g\,\boldsymbol{A[r, s+1:N]}\,\boldsymbol{u_S[s+1:N]}$
4:           $\boldsymbol{A[r, s+1:N]}$ -= $vr_s\,\boldsymbol{u_S^T[s+1:N]}$
5:       $\boldsymbol{v_s^T[s:N]} = g\,\boldsymbol{u_S^T[s+1:N]}\,\boldsymbol{A[s+1:N, s:N]}$
6:       for $r = s$ to $N$ do
7:           $\boldsymbol{A[r, s:N]}$ -= $\boldsymbol{u_S}[r]\,\boldsymbol{v_s^T[s:N]}$
8:           $vr_s = g\,A[r, s+1:N]\,u_S[s+1:N]$
9:           $A[r, s+1:N]$ -= $vr_s\,u_S^T[s+1:N]$
10:  end SPH
```

4. Results and Discussions

The experiments were carried out on the Volgograd State Technical University (VSTU) computing cluster. Currently, the cluster includes 24 nodes, including 16 nodes based on Intel Xeon of different generations, 8 nodes (2 servers) based on Intel Xeon Phi of the 3rd generation (KNL), 324×86 CPU cores, up to 32 Xeon Phi KNC accelerators, 4 NVidia TESLA V100/P100/K20 accelerators, the total amount of RAM is 4 TB, the Infiniband 56 Gbit network is used, and peak performance is more than 80 GFlops. Linux CentOS 7 is installed on most nodes.

We used RAPL in experiments aimed at determining software energy efficiency. RAPL is an interface developed by Intel, which provides built-in counters of CPU energy consumption. The measurement accuracy depends on the actual microarchitecture, but the developers claim that, starting with the Haswell architecture, it is based on fairly accurate measurements. It should be noted that RAPL is not an analog tool, an energy consumption software model is used. This model estimates energy consumption using hardware performance counters.

Only the CPU and Dynamic Random Access Memory (DRAM) are covered by measurements in most systems, so it is difficult to answer the question of how much energy the entire system consumes using RAPL. However, despite this, this interface is an excellent tool designed for monitoring energy consumption and optimizing applications from the energy efficiency point of view.

RAPL provides per-package energy estimates, meaning the totals are at the socket (not per-core) level. Energy measurements available include: total package, Power Plane 0 (PP0) which is the aggregate total of all cores, Power Plane 1 (PP1) which is an implementation-defined part of the "uncore" (usually the Graphics Processing Unit, GPU), and the DRAM. Measurement availability varies by chip model. DRAM measurements were originally only available for server systems but, starting with Haswell, are available on all CPUs. Similarly, GPU measurements are not available on server versions of the CPUs. A number of publications present RAPL interface studies on various architectures, and fairly high measurement accuracy is noted.

During the experiments, the software execution time and energy consumed by the computing system were measured. The results are shown in Figures 5 and 6.

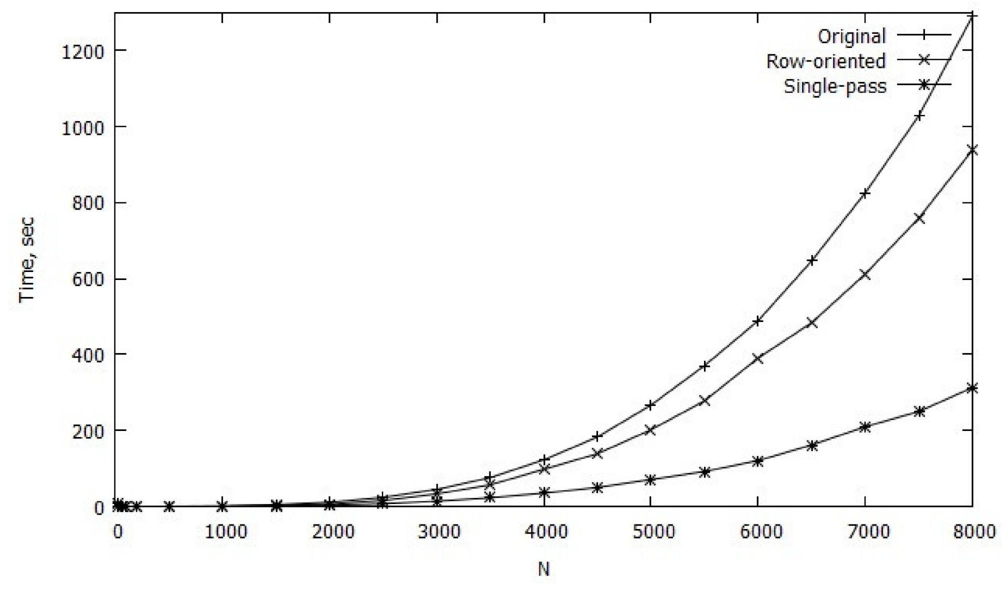

Figure 5. Dependence of the software execution time on the problem dimension.

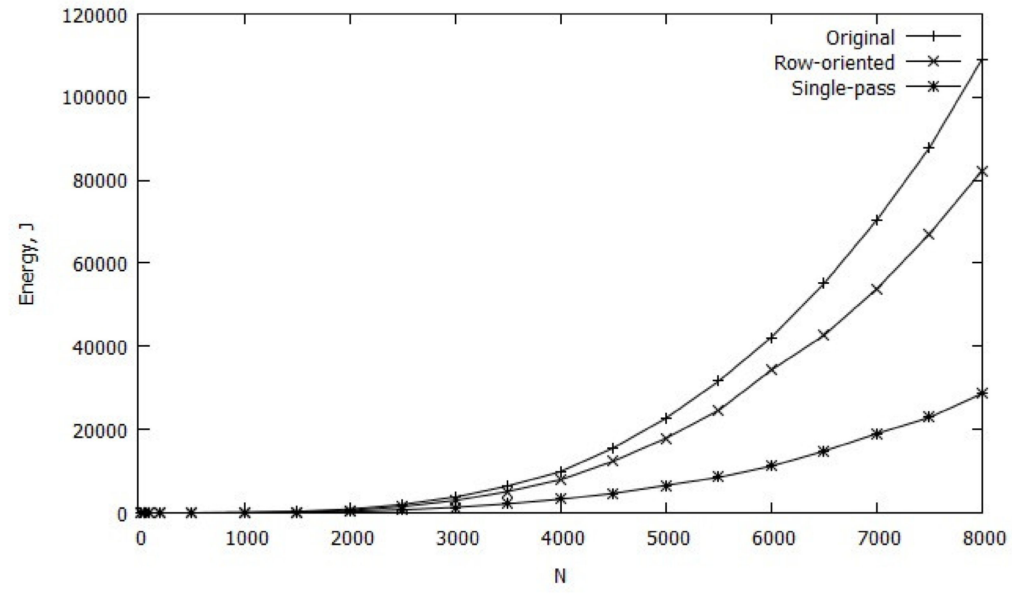

Figure 6. Dependence of the energy consumed on the problem dimension.

As expected, the single-pass transformation scheme turned out to be the fastest, and the implementation of the classic two-sided Householder transformation scheme was the slowest.

The energy consumption graphs turned out to be similar to the software execution time graphs; the fastest scheme turned out to be the most energy-efficient, and the least—the slowest.

The experiments carried out confirmed the hypothesis expressed in some research papers [22,23]. The fastest program turned out to be the most energy efficient. For example, when processing matrices of dimension 8000, the single-pass scheme implementation led to a consumption of 28.7 kJ, while the classical transformation scheme implementation on the same dimension led to a consumption of 109 kJ, which is 3.8 times more. Thus, it can be argued that any optimization aimed at reducing software execution time also leads to an increase in software energy efficiency. Additionally, although optimized versions can consume more power, energy consumption is reduced. At the same time, a direct relationship between the execution time reduction and power consumption during various optimizations is unlikely to be observed.

The answer to the question, "What is the quantitative relationship between the software execution time reduction and the energy consumption decreasing?" is not so obvious. Figures 7 and 8 show graphs of the power consumed in the software execution process.

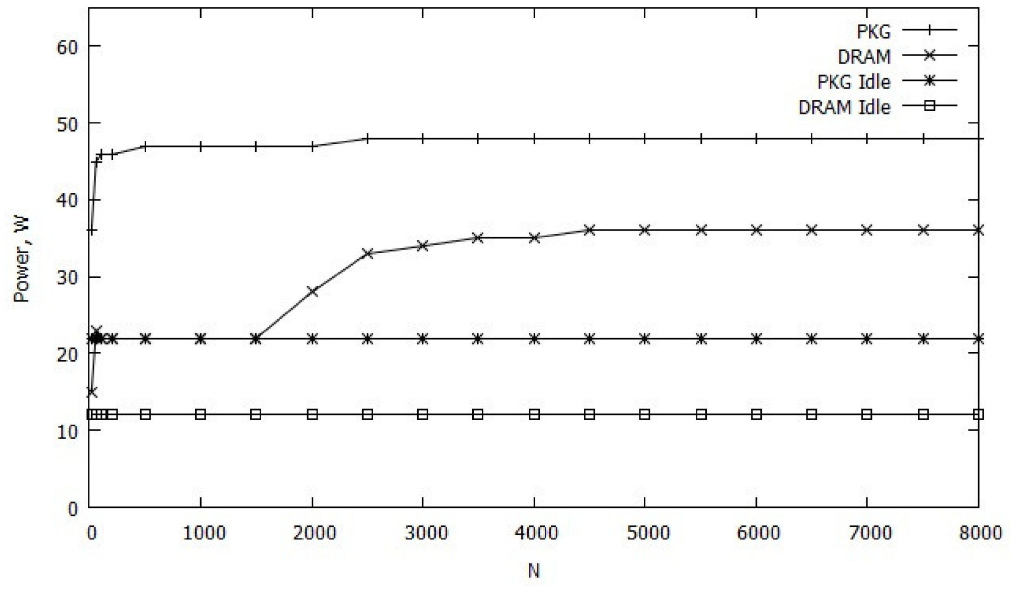

Figure 7. The dependence of the power consumed in the software execution process on the problem dimension (classical scheme).

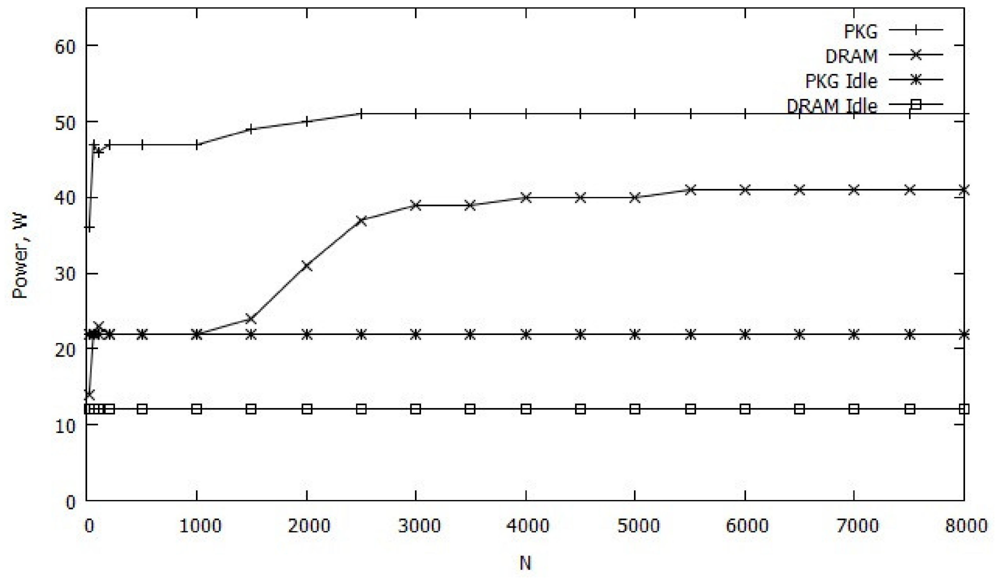

Figure 8. The dependence of the power consumed in the software execution process on the problem dimension (single-pass scheme).

Both graphs have the same structure and they present the following data:

PKG—the power consumed by the socket as a whole (Package);
DRAM—the power consumed by DRAM;
PKG Idle—the power consumed by the socket in «idle» mode;
DRAM Idle—the power consumed by DRAM in «idle» mode.

These figures confirm previously made assumptions that optimized software versions consume more power. So, when implementing the classical scheme, 49 W is consumed versus 52 W in the single-pass scheme (domain PKG) and 37 W versus 41 W (domain DRAM). However, due to a significant reduction in software execution time, energy consumption is reduced.

The "idle" mode here refers to the system operating mode without running user software. In this mode, some system processes are executed that do not actually consume resources (according to the data of the "top" utility). Analyzing Figures 7 and 8, it is obvious that in the "idle" mode, the CPU consumes power of 22 W and 12.5 W for the PKG and DRAM domains, respectively. It should also be noted that the graphs in Figure 6 are obtained by summing the values of PKG and DRAM.

Power consumed measurements in the "idle" mode were carried out as follows. As mentioned above, we used one of the VSTU computing cluster nodes for computational experiments. This node contains two Intel Xeon E5-2650 v3 2.3 GHz CPUs, and 128 Gb RAM. The software measuring the power consumption was run on one CPU, and the second remained in "idle" mode. The RAPL provides an opportunity to find out the number of CPUs available to the interface. In terms of RAPL, several package power domains can be controlled using RAPL. This is equal to the CPU package number in the system. Then, a set of parameters can be obtained for each CPU.

MSR (Model-Specific Registers) are used to obtain data. MSR are any of the various control registers in the ×86 instruction set used for debugging, program execution tracing, computer performance monitoring, and toggling certain CPU features. Reading and writing to these registers are handled by the *rdmsr* and *wrmsr* instructions, respectively. As these are privileged instructions, they can be executed only by the operating system. Use of the Linux MSR Kernel Module creates a pseudo file "/dev/cpu/x/msr" (with a unique x for each CPU or CPU core). A user with permissions to read and/or write to this file can use the file Input/Output Application Programming Interface (I/O API) to access these registers. The purpose of registers may vary from model to model.

Only two of all RAPL domains are supported on Intel Xeon E5 v3 CPUs—PKG and DRAM. These domains are presented on the graphs in the "idle" mode and the software execution process. Since the experiments were carried out in the CentOS environment, we used the file I/O API to read the contents of the registers. Figure 9 shows the pseudo-code of MSR reading.

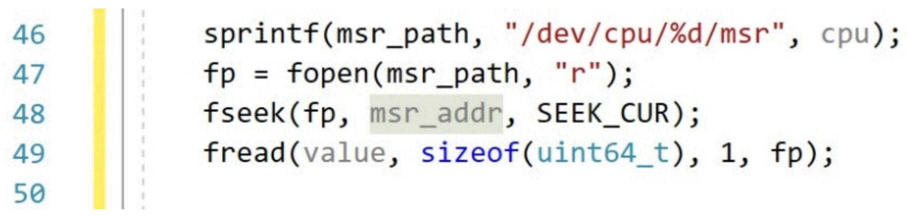
```
sprintf(msr_path, "/dev/cpu/%d/msr", cpu);
fp = fopen(msr_path, "r");
fseek(fp, msr_addr, SEEK_CUR);
fread(value, sizeof(uint64_t), 1, fp);
```

Figure 9. Pseudo-code of MSR reading.

Here, msr_addr is the required MSR. In particular, the RAPL PKG energy consumed register and the RAPL DRAM energy consumed register. This MSR is updated approximately every 1 msec. So, short workloads will have significant errors due to the refresh rate. In this regard, in the parameter measuring process, the calculations were repeated many times with next averaging of the results. The number of repetitions was set large enough for small dimensions to overcome the limitations associated with the frequency of MSR updates.

We constructed several graphs to assess the quantitative relationship between the software execution time reduction and energy consumption decreasing. So, Figure 10 shows the speedup of the optimized software relative to the classical transformation scheme. To construct this dependence, the data presented in Figure 5 were used.

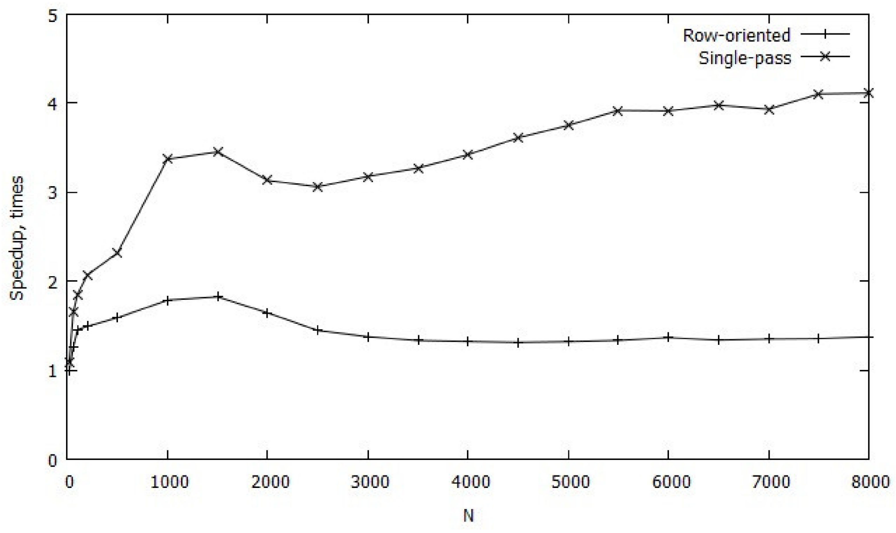

Figure 10. The speedup of the optimized software relative to the classical transformation scheme.

Both implementations show good performance growth at dimensions 1000–1500, which then decreases somewhat. These results are related to the CPU characteristics, specifically with the size of the cache memory, which for the used CPU is 25 Mb (L3 Smart Cache). The most speedup is achieved at dimensions of approximately 1750 with this cache memory amount and the double type (the length is 8 bytes), which we used for modeling. This is due to the fact that larger-dimensional matrices no longer fit into the cache, optimized versions increase acceleration on smaller dimensions due to better use of caches L1/L2. At dimensions greater than 1750, all three versions begin to generate misses in L3, which somewhat reduces the acceleration value. Further, the performance drops somewhat and, for a row-oriented scheme, the speedup stabilizes at around 1.35. For a single-pass scheme, again there is an increase in performance up to a value of 4.1 at dimension 8000. The performance gain, in this case, is provided by an efficient data caching algorithm for L2, while the basic scheme generates an increasing number of cache misses.

Next, the ratio of energy consumption of the basic and single-pass transformation scheme was calculated (Figure 11). Figure 6 data were used for the calculation (the ratio of the total energy for the PKG and DRAM domains calculated for the basic and single-pass transformation scheme). In general, the graph is like the speedup graph for a single-pass scheme, slightly differing in specific values.

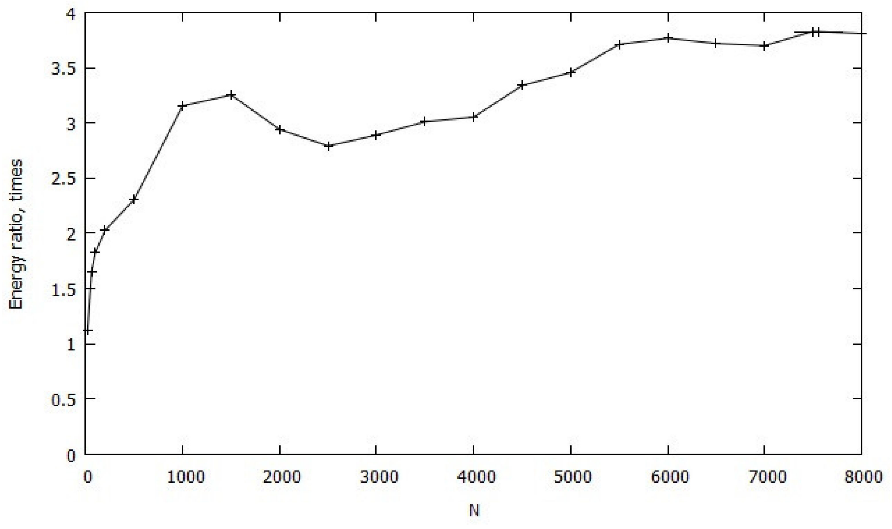

Figure 11. The ratio of the energy consumption of the basic and single-pass transformation scheme.

The dependency shown in Figure 12 illustrates this difference. The ratio of the speedup value to the power consumption decreasing value for a single-pass scheme is shown here. It can be seen that this ratio is slightly higher than one, and varies between 1.05–1.1 for different dimensions.

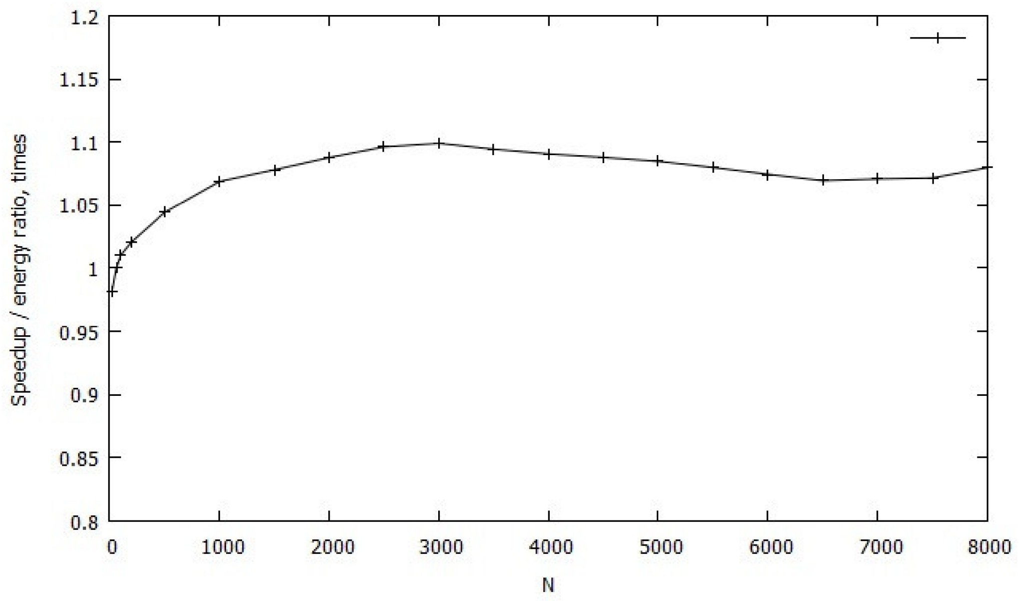

Figure 12. The ratio of speedup to the decreasing of energy consumption for a single-pass scheme.

That is, the absolute speedup value for a particular dimension slightly exceeds the reducing energy consumption value, although these values are close. In the course of the experiments, an assessment was also made of the "extra energy consumption" reduction, which we can define as the difference between current energy consumption and energy consumption in the "idle" mode (5).

$$E_{diff} = (PKG + DRAM) - (PKG\ Idle + DRAM\ Idle) \tag{5}$$

However, the analysis showed that these parameter dynamics for the target computing architecture almost completely coincide with the whole energy consumption dynamics. In addition, the graphs of the ratio of "extra energy consumption" and whole energy consumption practically coincide for the two studied software. In fact, let w_1 and w_2—"extra" power consumed by two software, and K = (*PKG Idle* + *DRAM Idle*)—the power consumed by the CPU in the "idle" mode. Then

$$\frac{w_1}{w_2} - \frac{(K + w_1)}{(K + w_2)} = \frac{(w_1/w_2 - 1)}{(1 + w_2/K)} \tag{6}$$

Earlier (see Figures 7 and 8) it was shown that the difference between the power consumption for the basic and single-pass schemes is not so great (does not exceed 10%). Accordingly, the value of the Expression (6) that shows the difference between the ratio of "extra" and total power consumption for the target computing system is small.

5. Conclusions

Currently, improving the energy efficiency of any activity type, whether it is industrial production or scientific research, engineering calculations, or the use of household appliances is becoming more relevant than ever. All this applies equally to IT technologies and energy-efficient software development. So, in this article, we conducted a study related to the assessment of the relationship between the proposed software optimization method (SCO) with an increase in software energy efficiency.

The obtained results show a high correlation degree between the decrease in software execution time and power consumption. The paper demonstrated the use of the application cache optimization technique SCO, which resulted in a significant reduction in software execution time. At certain dimensions, the optimized software version gives a speedup of more than 4 times in relation to the basic implementation of the transformation. At the same time, we measured the energy consumed using the RAPL interface provided by Intel for calculating the energy consumed during the software execution. We carried out measurements for the obtained optimized versions of the basic algorithm. Separately, we analyzed the values of the energy consumed by the CPU socket, and DRAM both during the software execution and in the «idle» mode. Measurements have shown that optimized algorithm versions reduce power consumption up to 4 times in relation to the basic transformation scheme. These measurements showed the effectiveness of the SCO-based methodology used to reduce energy consumption.

Unlike the heuristic approach to improving caching algorithms, SCO provides a formal tool for optimization. This can reduce the time for software modification, and give a certain economic effect by reducing energy consumption, computing resources and some other factors. There are no limitations on the use of this methodology, it can be applied to any software. However, its use can give a significant result in the case of tasks with regular access and a large amount of processed data.

The result of further work in this direction will be attempts to determine the impact of other software optimization methods on energy consumption, which would help developers create better software in terms of energy efficiency.

Author Contributions: Conceptualization, A.G.K. and V.E.; Investigation, V.E.; Methodology, A.G.K. and V.E.; Software, V.E.; Visualization, V.E.; Writing—original draft, V.E.; Writing—review & editing, A.G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fallis, E.; Spachos, P. Power Consumption and Throughput of Wireless Communication Technologies for Smartphones. In Proceedings of the 2018 Global Information Infrastructure and Networking Symposium (GIIS), Thessaloniki, Greece, 23–25 October 2018; pp. 1–4. [CrossRef]
2. Lutui, P.R.; Cusack, B.; Maeakafa, G. Energy efficiency for IoT devices in home environments. In Proceedings of the 2018 IEEE International Conference on Environmental Engineering (EE), Milan, Italy, 12–14 March 2018; pp. 1–6. [CrossRef]
3. Schill, A.; Globa, L.; Stepurin, O.; Gvozdetska, N.; Prokopets, V. Power Consumption and Performance Balance (PCPB) scheduling algorithm for computer cluster. In Proceedings of the 2017 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo), Odessa, UKraine, 11–15 September 2017; pp. 1–8. [CrossRef]
4. Rahmati, A.; Qian, A.; Zhong, L. Understanding human-battery interaction on mobile phones. In Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services, Singapore, 9–12 September 2007.
5. Zakarya, M.; Gillam, L. Energy efficient computing, clusters, grids and clouds: A taxonomy and survey. In *Sustainable Computing: Informatics & Systems*; Elsevier: Amsterdam, The Netherlands, 2017.
6. Zakarya, M. Energy, performance and cost efficient datacenters: A survey. In *Renewable and Sustainable Energy Reviews*; Elsevier: Amsterdam, The Netherlands, 2018.
7. Iyer, S.G.; Dipakumar Pawar, A. GPU and CPU Accelerated Mining of Cryptocurrencies and their Financial Analysis. In Proceedings of the 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 30–31 August 2018; pp. 599–604. [CrossRef]
8. Pereira, R.; Couto, M.; Ribeiro, F.; Rua, R.; Cunha, J.; Fernandes, J.P.; Saraiva, J. Energy Efficiency across Programming Languages. In Proceedings of the SLE'17, Vancouver, BC, Canada, 23–24 October 2017; 12p. [CrossRef]
9. Pandey, N.; Verma, O.P.; Kumar, A. A framework for usage pattern–based power optimization and battery lifetime prediction in smartphones. *Pers. Ubiquit. Comput.* **2022**, *26*, 821–836. [CrossRef]
10. Yang, R.; Song, J.; Huang, B.; Li, W.; Qi, G. An Energy-Efficient Step-Counting Algorithm for Smartphones. *Comput. J.* **2020**, *65*, 689–700. [CrossRef]

11. Oliveira, W.; Oliveira, R.; Castor, F. A study on the energy consumption of Android app development approaches. In Proceedings of the 14th International Conference on Mining Software Repositories, Buenos Aires, Argentina, 20–21 May 2017; pp. 42–52.
12. Li, D.; Halfond, W.G.J. An investigation into energy-saving programming practices for Android smartphone app development. In Proceedings of the 3rd International Workshop on Green and Sustainable Software, Hyderabad, India, 1 June 2014.
13. Sahin, C.; Cayci, F.; Gutierrez, I.L.M.; Clause, J.; Kiamilev, F.; Pollock, L.; Winbladh, K. Initial explorations on design pattern energy usage. In Proceedings of the 4th International Workshop on Green and Sustainable Software, Zurich, Switzerland, 3 June 2012; pp. 55–61.
14. Chen, Y.-J.; Hsu, C.-H.; Li, K.-C.; Chang, H.-Y.; Wang, S.-T. Power Consumption Optimization of MPI Programs on Multi-core Clusters. In *Scalable Information Systems*; Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, INFOSCALE 2009; Mueller, P., Cao, J.N., Wang, C.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 18. [CrossRef]
15. Görtz, M.D.; Kühn, R.; Zietek, O.; Bernhard, R.; Bulinski, M.; Duman, D.; Freisen, B.; Jentsch, U.; Klöppner, T.; Popovic, D.; et al. Energy Efficiency of a Low Power Hardware Cluster for High Performance Computing. In *INFORMATIK 2017*; Eibl, M., Gaedke, M., Eds.; Gesellschaft für Informatik: Bonn, Germany, 2017; pp. S2537–S2548. [CrossRef]
16. Pereira, R.; Carção, T.; Couto, M.; Cunha, J.; Fernandes, J.P.; Saraiva, J. Helping programmers improve the energy efficiency of source code. In Proceedings of the 39th International Conference on Software Engineering—Companion, ICSE-C, Buenos Aires, Argentina, 20–28 May 2017.
17. Chowdhury, S.A.; Hindle, A. Greenoracle: Estimating software energy consumption with energy measurement corpora. In Proceedings of the 13th International Conference on Mining Software Repositories, MSR, Austin, TX, USA, 14–15 May 2016; pp. 49–60.
18. Sahin, C.; Pollock, L.; Clause, J. How do code refactorings affect energy usage? In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy, 18–19 September 2014; p. 36.
19. Hao, S.; Li, D.; Halfond, W.G.J.; Govindan, R. Estimating mobile application energy consumption using program analysis. In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, 18–26 May 2013; pp. 92–101.
20. Couto, M.; Borba, P.; Cunha, J.; Fernandes, J.P.; Pereira, R.; Saraiva, J. Products go green: Worst-case energy consumption in software product lines. In Proceedings of the 21st International Systems and Software Product Line Conference, SPLC, Sevilla, Spain, 25–29 September 2017; Volume A, pp. 84–93.
21. Pereira, R.; Couto, M.; Saraiva, J.; Cunha, J.; Fernandes, J.P. The influence of the Java collection framework on overall energy consumption. In Proceedings of the 5th International Workshop on Green and Sustainable Software, Austin, TX, USA, 14–22 May 2016; pp. 15–21.
22. Yuki, T.; Rajopadhye, S. Folklore confirmed: Compiling for speed = compiling for energy. In *Languages and Compilers for Parallel Computing*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 169–184.
23. Pinto, G.; Castor, F.; Liu, Y.D. Understanding energy behaviors of thread management constructs. In Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications, Portland, OR, USA, 20–24 October 2014; pp. 345–360.
24. Trefethen, A.E.; Thiyagalingam, J. Energy-aware software: Challenges, opportunities and strategies. *J. Comput. Sci.* **2013**, *4*, 444–449. [CrossRef]
25. Lima, L.G.; Melfe, G.; Soares-Neto, F.; Lieuthier, P.; Fernandes, J.P.; Castor, F. Haskell in green land: Analyzing the energy behavior of a purely functional language. In Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER, Suita, Japan, 14–18 March 2016; pp. 517–528.
26. Abdulsalam, S.; Zong, Z.; Gu, Q.; Qiu, M. Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency. In Proceedings of the 6th International Green and Sustainable Computing Conference, IGCC, Las Vegas, NV, USA, 14–16 December 2015; pp. 1–8.
27. Ortega, J.M. *Introduction to Parallel and Vector Solution of Linear Systems*; Plenum Press: New York, NY, USA, 1988; p. 305. ISBN 0-306-42862-8.
28. Demmel, J.W. *Applied Numerical Linear Algebra*; SIAM: Philadelphia, PA, USA, 1997; p. 430.
29. Salam, A.; Kahla, H.B. An upper J-Hessenberg reduction of a matrix through symplectic Householder transformations. *Comput. Math. Appl.* **2019**, *78*, 178–190. [CrossRef]
30. Bujanovic, Z.; Karlsson, L.; Kressner, D. A householder-based algorithm for Hessenberg-triangular reduction. *SIAM J. Matrix Anal. Appl.* **2018**, *39*, 1270–1294. [CrossRef]
31. Bogoya, J.M.; Grudsky, S.M.; Malysheva, I.S. Extreme individual eigenvalues for a class of large hessenberg toeplitz matrices. *Oper. Theory Adv. Appl.* **2018**, *271*, 119–143.
32. Kabir, K.; Haidar, A.; Tomov, S.; Dongarra, J. Performance analysis and design of a hessenberg reduction using stabilized blocked elementary transformations for new architectures. *Simul. Ser.* **2015**, *47*, 135–142.
33. Tomov, S.; Nath, R.; Dongarra, J. Accelerating the reduction to upper hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing. *Parallel Comput.* **2010**, *36*, 645–654. [CrossRef]
34. Buttari, A.; Langou, J.; Kurzak, J.; Dongarra, J. Parallel tiled QR factorization for multicore architectures. *Concurr. Comput. Pract. Exp.* **2008**, *20*, 1573–1590. [CrossRef]

35. Merchant, F.; Vatwani, T.; Chattopadhyay, A.; Nandy, S.K.; Narayan, R. Efficient realization of householder transform through algorithm-architecture co-design for acceleration of QR Factorization. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 1707–1720. [CrossRef]
36. Noble, J.H.; Lubasch, M.; Stevens, J.; Jentschura, U.D. Diagonalization of complex symmetric matrices: Generalized Householder reflections, iterative deflation and implicit shifts. *Comput. Phys. Commun.* **2017**, *221*, 304–316. [CrossRef]
37. Getmanskiy, V.; Andreev, A.; Alekseev, S.; Gorobtsov, A.; Egunov, V.; Kharkov, E. Optimization and Parallelization of CAE Software Stress-Strain Solver for Heterogeneous Computing Hardware. In Proceedings of the Creativity in Intelligent Technologies and Data Science. Second Conference, CIT&DS 2017, Volgograd, Russia, 12–14 September 2017; Communications in Computer and Information Science. Springer International Publishing: Berlin/Heidelberg, Germany, 2017; Volume 754, pp. 562–674.
38. Low, T.M.; Igual, F.D.; Smith, T.M.; Quintana-Orti, E.S. Analytical Modeling Is Enough for High-Performance BLIS. *ACM Trans. Math. Softw.* **2016**, *43*, 1–18. [CrossRef]
39. Schreiber, R.; VanLoan, C. A Storage-Efficient WY Representation for Products of Householder Transformations. *SIAM J. Sci. Stat. Comput.* **1989**, *10*, 53–57. [CrossRef]
40. Golub, G.; Van Loan, C. *Matrix Computations*, 3rd ed.; The Johns Hopkins University Press: Baltimore, MD, USA, 1996; p. 728.
41. Eljammaly, M.; Karlsson, L.; Kagstrom, B. On the tunability of a new Hessenberg reduction algorithm using parallel cache assignment. In *Parallel Processing and Applied Mathematics*; Springer: Cham, Switherland, 2018; Volume 10777, pp. 579–589.
42. Andreev, A.; Doukhnitch, E.; Egunov, V.; Zharikov, D.; Shapovalov, O.; Artuh, S. Evaluation of Hardware Implementations of CORDIC-Like Algorithms in FPGA Using OpenCL Kernels. In *Knowledge-Based Software Engineering, Proceedings of 11th Joint Conference, JCKBSE, Volgograd, Russia, 17–20 September 2014*; Communications in Computer and Information Science; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; Volume 466, pp. 228–242.
43. Egunov, V.A.; Kravets, A.G. Povyshenie effektivnosti programm dlya vychislitel'nyh sistem s ierarhicheskoj strukturoj pamyati [Improving the efficiency of software for computing systems with a hierarchical memory structure]. *Mat. Metod. V Tekhnologiyah I Tekhnike* **2022**, *4*, 100–103. [CrossRef]
44. Glinsky, B.; Kulikov, I.; Chernykh, I.; Weins, D.; Snytnikov, A.; Nenashev, V.; Andreev, A.; Egunov, V.; Kharkov, E. The Co-design of Astrophysical Code for Massively Parallel Supercomputers. In Proceedings of the Algorithms and Architectures for Parallel Processing, ICA3PP 2016 Collocated Workshops: SCDT, TAPEMS, BigTrust, UCER, DLMCS, Granada, Spain, 14–16 December 2016; Lecture Notes in Computer Science. Springer International Publishing: Berlin/Heidelberg, Germany, 2016; Volume 10049, pp. 342–353.
45. Dominguez, T.; Quintana, A.E.; Orti, E.S. Fast Blocking of Householder Reflectors on Graphics Processors. In Proceedings of the 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP, Cambridge, UK, 21–23 March 2018; pp. 385–393.
46. Egunov, V.A.; Andreev, A.E. Vektorizaciya algoritmov vypolneniya sobstvennogo i singulyarnogo razlozhenij matric s ispol'zovaniem preobrazovaniya Hauskholdera [Vectorization of algorithms for performing proper and singular matrix expansions using the Householder transformation]. *Prikaspijskij Zhurnal: Upr. I Vysok. Tekhnologii* **2020**, *2*, 71–85.
47. Andreev, A.E. Solving of Eigenvalue and Singular Value Problems via Modified Householder Transformations on Shared Memory Parallel Computing Systems. In *Supercomputing: RuSCDays, Proceedings of the 5th Russian Supercomputing Days, Moscow, Russia, 23–24 September 2019*; Springer: Cham, Switzerland, 2019; Volume 1129, pp. 131–151.
48. Egunov, V.A. Implementation of QR and LQ decompositions on shared memory parallel computing systems. In Proceedings of the 2nd International Conference on Industrial Engineering, Applications and Manufacturing, ICIEAM, Chelyabinsk, Russia, 19–20 May 2016; p. 5. [CrossRef]